HTC Job Execution with HTCondor

Tuesday, July 14

Lauren Michael
Overview

- How does the HTCondor job scheduler work?
- How do you run, monitor, and review jobs?
- Best ways to submit multiple jobs (what we’re here for, right?)
- Testing, tuning, and troubleshooting to scale up.
Example Local Cluster

- UW-Madison’s Center for High Throughput Computing (CHTC)
- Recent CPU hours:
  - ~120 million hrs/year (~13k cores)
  - up to 15,000 per user, per day (~600 cores in use)
HTCondor History and Status

• History
  – Started in 1988 as a “cycle scavenger”

• Today
  – Developed within the CHTC by professional developers
  – Used all over the world, by:
    ▪ Campuses, national labs, Einstein/Folding@Home
    ▪ Dreamworks, Boeing, SpaceX, investment firms, …
    ▪ The Open Science Grid!!

• Miron Livny,
  – Professor, UW-Madison Computer Sciences
  – CHTC Director, HTCondor PI, OSG Technical Director
HTCondor -- How It Works

- Submit tasks to a queue (on a submit server)
- HTCondor schedules them to run on computers (execute server)
Terminology: Job

- **Job**: An independently-scheduled unit of computing work

- Three main pieces:
  - **Executable**: the script or program to run
  - **Input**: any options (arguments) and/or file-based information
  - **Output**: any files or screen information produced by the executable

- In order to run *many* jobs, executable must run on the command-line without any graphical input from the user
Terminology: **Machine, Slot**

- **Machine**
  - A whole computer (desktop or server)
  - Has multiple processors (**CPU cores**), some amount of **memory**, and some amount of file space (**disk**)

- **Slot**
  - an assignable unit of a machine (i.e. 1 job per slot)
  - most often, corresponds to one core with some memory and disk
  - a typical machine will have multiple slots

- HTCondor can break up and create new slots, dynamically, as resources become available from completed jobs
Job Matching

- On a regular basis, the central manager reviews *Job* and *Machine* attributes and matches jobs to *Slots*. 
Job Execution

• (Then the submit and execute points communicate directly.)
Single Computer

execute

submit +
central manager

execute

execute

execute
BASIC JOB SUBMISSION
Job Example

- program called “compare_states” (executable), which compares two data files (input) and produces a single output file.

```bash
$ compare_states wi.dat us.dat wi.dat.out
```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
Basic Submit File

- List your `executable` and any `arguments` it takes

```bash
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- Arguments are any options passed to the executable from the command line

```bash
$ compare_states wi.dat us.dat wi.dat.out
```
Basic Submit File

executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1

• comma-separated list of input files to transfer to the slot
Basic Submit File

- HTCondor will transfer back all new and changed files (output) from the job, automatically.

```bash
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```
Basic Submit File

- **log**: file created by HTCondor to track job progress
  - *Explored in exercises!*

- **output/error**: captures stdout and stderr from your program (what would otherwise be printed to the terminal)

```plaintext
executable = compare_states
arguments = wi.dat us.dat wi.dat.out
transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB
queue 1
```
Basic Submit File

- **request** the resources your job needs.
  - *More on this later!*
- **queue**: keyword indicating “create 1 job”

```plaintext
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```
SUBMITTING AND MONITORING
Submitting and Monitoring

- To submit a job/jobs: `condor_submit submit_file`
- To monitor submitted jobs: `condor_q`

```
$ condor_submit job.submit
Submitting job(s).
1 job(s) submitted to cluster 128.

$ condor_q
-- Schedd: learn.chtc.wisc.edu : <128.104.101.92> @ 05/01/17 10:35:54
OWNER  BATCH_NAME             SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CMD: compare_states   5/9 11:05      _      _      1      1 128.0

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```
More about `condor_q`

- By default, `condor_q` shows your jobs only and batches jobs that were submitted together:

```
$ condor_q
-- Schedd: learn.chtc.wisc.edu : <128.104.101.92> @ 05/01/17 10:35:54
OWNER    BATCH_NAME         SUBMITTED   DONE   RUN   IDLE   TOTAL JOB_ID  
alice    CMD: compare_states 5/9   11:05   _     _     1      1 128.0

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

JobId = `ClusterId.ProcID`

- Limit `condor_q` by username, `ClusterId` or full `JobId`, (denoted `[U/C/J]` in following slides).
More about `condor_q`

- To see individual job details, use:
  
  ```bash
  condor_q -nobatch
  ```

  ```
  $ condor_q -nobatch
  -- Schedd: learn.chtc.wisc.edu : <128.104.101.92>
  ID   OWNER  SUBMITTED  RUN_TIME ST PRI SIZE  CMD
  128.0 alice  5/9 11:09  0+00:00:00 I  0   0.0 compare_states
  128.1 alice  5/9 11:09  0+00:00:00 I  0   0.0 compare_states
  ...
  
  1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
  ```

- We will use the `-nobatch` option in the following slides to see extra detail about what is happening with a job.
Job Idle

$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92>

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.0</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>compare_states wi.dat us.dat</td>
</tr>
</tbody>
</table>

1 jobs; 0 completed, 0 removed, 1 idle, running, 0 held, 0 suspended

Submit Node

(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
    job.log
    job.out
    job.err
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618>
<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.0</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0+00:00:00</td>
<td>0</td>
<td>0.0</td>
<td>compare_states wi.dat us.dat</td>
</tr>
</tbody>
</table>

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended

Submit Node

```
(submit_dir)/
  job.submit
  compare_states
  wi.dat
  us.dat
  job.log
  job.out
  job.err
```

Execute Node

```
(execute_dir)/
  compare_states
    wi.dat
    us.dat
```
$ condor_q -nobatch

-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92>

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.0</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0:01:08</td>
<td>R</td>
<td>0</td>
<td>0.0</td>
<td>compare_states wi.dat us.dat</td>
</tr>
</tbody>
</table>

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended

**Submit Node**

```
(submit_dir)/
  job.submit
  compare_states
  wi.dat
  us.dat
  job.log
  job.out
  job.err
```

**Execute Node**

```
(execute_dir)/
  compare_states
  wi.dat
  us.dat
  stderr
  stdout
  wi.dat.out
```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92>

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0+00:02:02</td>
<td>0.0</td>
<td>compare_states wi.dat us.dat</td>
</tr>
</tbody>
</table>

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
Job Completes (cont.)

```
$ condor_q -nobatch

-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618>...

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
</table>

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended

Submit Node

```

```
(submit_dir)/
  job.submit
  compare_states
  wi.dat
  us.dat
  job.log
  job.out
  job.err
  wi.dat.out
```
Job States

condor_submit

Idle (I) → Running (R) → Completed (C)

- In the queue:
  - transfer executable and input to execute node
  - in the queue

- Leaving the queue:
  - transfer output back to submit node
  - leaving the queue
000 (128.000.000) 05/09 11:09:08 Job submitted from host: <128.104.101.92&sock=6423_b881_3>
...
001 (128.000.000) 05/09 11:10:46 Job executing on host: <128.104.101.128:9618&sock=5053_3126_3>
...
006 (128.000.000) 05/09 11:10:54 Image size of job updated: 220
  1 - MemoryUsage of job (MB)
  220 - ResidentSetSize of job (KB)
...
005 (128.000.000) 05/09 11:12:48 Job terminated.
  (1) Normal termination (return value 0)
    Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage
  0 - Run Bytes Sent By Job
  33 - Run Bytes Received By Job
  0 - Total Bytes Sent By Job
  33 - Total Bytes Received By Job

Partitionable Resources: Usage Request Allocated

<table>
<thead>
<tr>
<th>Resource</th>
<th>Usage</th>
<th>Request</th>
<th>Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Disk (KB)</td>
<td>14</td>
<td>20480</td>
<td>17203728</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>
Resource Request

• Jobs are nearly always using a part of a machine (a single slot), and not the whole thing
• Very important to request appropriate resources (memory, cpus, disk)
  – requesting too little: causes problems for your and other jobs; jobs might be ‘held’ by HTCondor
  – requesting too much: jobs will match to fewer “slots” than they could, and you’ll block other jobs
# Is it OSG-able?

<table>
<thead>
<tr>
<th>Per-Job Resources</th>
<th>Ideal Jobs! (up to 10,000 cores, per user!)</th>
<th>Still Very Advantageous!</th>
<th>Probably not...</th>
</tr>
</thead>
<tbody>
<tr>
<td>cores (GPUs)</td>
<td>1 (1; non-specific)</td>
<td>&lt;8 (1; specific GPU type)</td>
<td>&gt;8 (or MPI) (multiple)</td>
</tr>
<tr>
<td>Walltime (per job)</td>
<td>&lt;10 hrs* *or checkpointable</td>
<td>&lt;20 hrs* *or checkpointable</td>
<td>&gt;20 hrs</td>
</tr>
<tr>
<td>RAM (per job)</td>
<td>&lt;few GB</td>
<td>&lt;10 GB</td>
<td>&gt;10 GB</td>
</tr>
<tr>
<td>Input (per job)</td>
<td>&lt;500 MB</td>
<td>&lt;10 GB</td>
<td>&gt;10 GB</td>
</tr>
<tr>
<td>Output (per job)</td>
<td>&lt;1 GB</td>
<td>&lt;10 GB</td>
<td>&gt;10 GB</td>
</tr>
<tr>
<td>Software</td>
<td>‘portable’ (pre-compiled binaries, transferable, containerizable, etc.)</td>
<td>most other than ☢☢☢</td>
<td>licensed software; non-Linux</td>
</tr>
</tbody>
</table>
000 (128.000.000) 05/09 11:09:08 Job submitted from host: <128.104.101.92&sock=6423_b881_3>

001 (128.000.000) 05/09 11:10:46 Job executing on host: <128.104.101.128:9618&sock=5053_3126_3>

006 (128.000.000) 05/09 11:10:54 Image size of job updated: 220

1 - MemoryUsage of job (MB)
220 - ResidentSetSize of job (KB)

005 (128.000.000) 05/09 11:12:48 Job terminated.

(1) Normal termination (return value 0)

Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage

0 - Run Bytes Sent By Job
33 - Run Bytes Received By Job
0 - Total Bytes Sent By Job
33 - Total Bytes Received By Job

<table>
<thead>
<tr>
<th>Partitionable Resources</th>
<th>Usage</th>
<th>Request</th>
<th>Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Disk (KB)</td>
<td>14</td>
<td>20480</td>
<td>17203728</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>
SUBMITTING MULTIPLE JOBS
From one job ...

```
job.submit

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

log = job.log
output = job.out
error = job.err

queue

(submit_dir)/

analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- Goal: create 3 jobs that each analyze a different input file.
One submit file per job
(not recommended!)

```
job0.submit
executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in
output = job0.out
error = job0.err
queue

job1.submit
executable = analyze.exe
arguments = file1.in file1.out
transfer_input_files = file1.in
output = job1.out
error = job1.err
queue
```

(submit_dir)/
```
analyze.exe
file0.in
file1.in
file2.in
(etc.)

job0.submit
job1.submit
job2.submit
(etc.)
```

(etc...)

Automatic Variables

Each job’s \texttt{ClusterId} and \texttt{ProcId} numbers are autogenerated and saved as job attributes.

You can reference them inside the submit file using:

- `$(ClusterId)`
- `$(ProcId)`

* `$(ClusterId)` and `$(ProcId)` are also okay
Using $(Process) for Numbered Files

```bash
job.submit

executable = analyze.exe
arguments = file$(Process).in file$(Process).out
transfer_input_files = file$(Process).in

log = job_$(_Cluster).log
output = job_$(_Process).out
error = job_$(_Process).err

queue 3
```

$(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- $(Process) and $(Cluster) allow us to provide unique values to each job and submission!
Organizing Files in Sub-Directories

- Create sub-directories and use paths in the submit file to separate various input, error, log, and output files.
Use a Directory* per File Type

(submit_dir)/

<table>
<thead>
<tr>
<th>job.submit</th>
<th>file0.out</th>
<th>input/</th>
<th>log/</th>
<th>err/</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyze.exe</td>
<td>file1.out</td>
<td>file0.in</td>
<td>job0.log</td>
<td>job0.err</td>
</tr>
<tr>
<td></td>
<td>file2.out</td>
<td>file1.in</td>
<td>job1.log</td>
<td>job1.err</td>
</tr>
<tr>
<td></td>
<td></td>
<td>file2.in</td>
<td>job2.log</td>
<td>job2.err</td>
</tr>
</tbody>
</table>

job.submit

- executable = analyze.exe
- arguments = file$(Process).in file$(Process).out
- transfer_input_files = input/file$(Process).in

- log = log/job$(Process).log
- error = err/job$(Process).err
- queue 3

*directories must be created before jobs are submitted
File always get transferred into the *top level* of the execute directory, regardless of how they are organized on the submit server.
Separating jobs with `InitialDir`

<table>
<thead>
<tr>
<th>(submit_dir)/</th>
<th>job0/</th>
<th>job1/</th>
<th>job2/</th>
</tr>
</thead>
<tbody>
<tr>
<td>job.submit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>analyze.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.in</td>
<td>file.in</td>
<td>file.in</td>
<td>file.in</td>
</tr>
<tr>
<td>job.log</td>
<td>job.log</td>
<td>job.log</td>
<td>job.log</td>
</tr>
<tr>
<td>job.err</td>
<td>job.err</td>
<td>job.err</td>
<td>job.err</td>
</tr>
<tr>
<td>file.out</td>
<td>file.out</td>
<td>file.out</td>
<td>file.out</td>
</tr>
</tbody>
</table>

`job.submit`:

- `executable = analyze.exe`
- `initialdir = job$(Process)`
- `arguments = file.in file.out`
- `transfer_input_files = file.in`
- `log = job.log`
- `error = job.err`
- `queue 3`

*directories must be created before jobs are submitted*
What about non-numbered jobs?

- Back to our `compare_states` example...
- What if we had data for each state? We could do 50 submit files (or 50 “queue 1” statements) ...

<table>
<thead>
<tr>
<th>executable</th>
<th>arguments</th>
<th>dat.out</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compare_states</code></td>
<td><code>vt.dat us.dat vt.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>al.dat us.dat al.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>tx.dat us.dat tx.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>ut.dat us.dat ut.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>ak.dat us.dat ak.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>tn.dat us.dat tn.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>sd.dat us.dat sd.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>mn.dat us.dat mn.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>vt.dat us.dat vt.dat.out</code></td>
<td>...</td>
</tr>
<tr>
<td><code>compare_states</code></td>
<td><code>ak.dat us.dat ak.dat.out</code></td>
<td>...</td>
</tr>
</tbody>
</table>
# Submitting Multiple Jobs – Queue Statements

<table>
<thead>
<tr>
<th>Multiple Submit Files (Multiple Queue Statements)</th>
<th>Not Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Var matching pattern</strong></td>
<td></td>
</tr>
<tr>
<td>queue state matching * .dat</td>
<td></td>
</tr>
<tr>
<td>queue directory matching job*</td>
<td></td>
</tr>
<tr>
<td><strong>Var in (i ii iii ...)</strong></td>
<td></td>
</tr>
<tr>
<td>queue state in (wi.dat ca.dat co.dat)</td>
<td></td>
</tr>
<tr>
<td><strong>Var1, Var2 from csv_file</strong></td>
<td></td>
</tr>
<tr>
<td>queue state from state_list.txt</td>
<td>wi.dat</td>
</tr>
<tr>
<td></td>
<td>ca.dat</td>
</tr>
<tr>
<td></td>
<td>mo.dat</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

**state_list.txt:**

<table>
<thead>
<tr>
<th>wi.dat</th>
<th>ca.dat</th>
<th>mo.dat</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Multiple Job Use Cases – Queue Statements

<table>
<thead>
<tr>
<th>multiple submit files</th>
<th><strong>Not recommended.</strong> Though, may be useful for separating job batches, conceptually, for yourself.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>var matching pattern</em></td>
<td>Minimal preparation, can use “files” or “dirs” keywords to narrow possible matches. Requires good naming conventions, less reproducible.</td>
</tr>
<tr>
<td><em>var in (i,ii,iii,…)</em></td>
<td><strong>All information contained in the submit file:</strong> reproducible. Harder to automate submit file creation.</td>
</tr>
<tr>
<td><em>var1,var2 from csv_file</em></td>
<td><strong>Supports multiple variables,</strong> highly modular (easy to use one submit file for many job batches that have different <em>var</em> lists), reproducible. Additional file needed, but can be automated.</td>
</tr>
</tbody>
</table>
TESTING AND TROUBLESHOOTING
What Can Go Wrong?

- Jobs can go wrong “internally”: the executable experiences an error
- Jobs can go wrong *logistically*, from HTCondor’s perspective:
  - a job can’t be matched
  - files not found for transfer
  - job used too much memory
  - badly-formatted executable
  - and more...
Reviewing Failed Jobs

- Job log, output and error files can provide valuable troubleshooting details:

<table>
<thead>
<tr>
<th>Log</th>
<th>Output</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>• when jobs were submitted, started, held, or stopped</td>
<td>• stdout (or other output files)</td>
<td>• stderr captures errors from the operating system, or reported by the executable, itself.</td>
</tr>
<tr>
<td>• where job ran</td>
<td>• any “print” or “display” information from your program (may contain errors from the executable)</td>
<td></td>
</tr>
<tr>
<td>• resources used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• interruption reasons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• exit status</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Job Holds

- HTCondor will *hold* your job if there’s logistical issue that YOU (or maybe an admin) need to fix.
  - files not found for transfer, over memory, etc.
- A job that goes on hold is interrupted (all progress is lost), but remains in the queue in the “H” state until removed, or (fixed and) released.
Diagnosing Holds

• If HTCondor puts a job on hold, it provides a hold reason, which can be viewed in the log file, with \texttt{condor\_q -hold <Job.ID>}, or with:

\begin{verbatim}
condor_q -hold -af HoldReason
\end{verbatim}

\begin{verbatim}
$ condor_q -hold -af HoldReason
Error from slot1_1@wid-003.chtc.wisc.edu: \textit{Job has gone over memory limit of 2048 megabytes.}
Error from slot1_20@e098.chtc.wisc.edu: \textit{SHADOW at 128.104.101.92 failed to send file(s) to <128.104.101.98:35110>: error reading from /home/alice/script.py: (errno 2) No such file or directory; STARTER failed to receive file(s) from <128.104.101.92:9618>}
Error from slot1_11@e138.chtc.wisc.edu: \textit{STARTER at 128.104.101.138 failed to send file(s) to <128.104.101.92:9618>; SHADOW at 128.104.101.92 failed to write to file /home/alice/Test_18925319_16.err: (errno 122) Disk quota exceeded}
\end{verbatim}
Common Hold Reasons

- Job has used **more memory or disk** than requested.
- Incorrect path to files that need to be transferred.
- **Badly formatted executables**
  (e.g. Windows line endings on Linux)
- Submit directory is **over quota**.
- Job has run for **too long**.
  (72-hour default in CHTC Pool)
- The **admin has put your job on hold**.
Holding and Removing Jobs

- If you know your job has a problem and it hasn’t yet completed, you can fix it!
- If the problem requires resubmission:
  - Remove it from the queue:
    \texttt{condor\_rm [U/C/J]}
- If problem is within the executable or input file(s):
  - Hold the job, fix it, and release it:
    \texttt{condor\_hold [U/C/J]}
    \texttt{condor\_release [U/C/J]}
YOUR TURN!

CHTC Pool

- single-core
- high-memory
- multi-core
- GPUs
- MPI

submit
server
Thoughts on Exercises

• Copy-and-paste is quick, but you *WILL* learn more by typing out commands and submit file contents
• Ask Questions during Work Time! (Slack)
• Exercises in THIS unit are important to finish before moving on! (You can save “bonus” exercises for later.)

• (See 1.6 if you need to remove jobs!)
Reviewing Jobs

• To review a large group of jobs at once, use `condor_history`

As `condor_q` is to the present, `condor_history` is to the past

```bash
$ condor_history alice

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>COMPLETED</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>189.1012</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:07:37</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.1002</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:08:03</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.1081</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:03:16</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.944</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:11:15</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.659</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:26:56</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.653</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:27:07</td>
<td>C</td>
<td>5/11 16:00</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.1040</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:05:15</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.1003</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:07:38</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.962</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:09:36</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.961</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:09:43</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
<tr>
<td>189.898</td>
<td>alice</td>
<td>5/11 09:52</td>
<td>0+00:13:47</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
</tbody>
</table>
```
Using Multiple Variables

• Both the “from” and “in” syntax support multiple variables from a list.

```
job.submit

executable = compare_states
arguments = \-y \$(year) \-i \$(infile)

transfer_input_files = \$(infile)

queue infile,year from job_list.txt
```

```
job_list.txt

wi.dat, 2010
wi.dat, 2015
ca.dat, 2010
ca.dat, 2015
mo.dat, 2010
mo.dat, 2015
```
Shared Files

- HTCondor can transfer an entire directory or all the contents of a directory
  - transfer whole directory
    
    ```
    transfer_input_files = shared
    ```
  - transfer contents only
    
    ```
    transfer_input_files = shared/
    ```

- Useful for jobs with many shared files; transfer a directory of files instead of listing files individually

```python
job.submit
shared/
  reference.db
  parse.py
  analyze.py
  cleanup.py
  links.config
```

OSG Virtual School Pilot 2020