An Introduction to Using HTCondor

HTCondor Week 2022
Christina Koch
Covered In This Tutorial

• What is HTCondor?
• Set up and Run a Job with HTCondor
• Submit Multiple Jobs with HTCondor
(pause for questions)

• How HTCondor Matches and Runs Jobs
• Testing and Troubleshooting
• Use Cases and HTCondor Features
• Automation
Introduction
What is HTCondor?

• Software that schedules and runs computing tasks on computers
How It Works

- Submit tasks to a queue (on an Access Point)
- HTCondor schedules them to run on computers (Execute Points)
HTCondor on Many Computers

Access Point
queue

execute
execute
execute

tasks
Why HTCondor?

• HTCondor manages and runs work on your behalf.
• Manage shared resources among users:
  • Schedule tasks on a single computer to manage computer capacity.
  • Schedule tasks on a group* of computers (which may/may not be directly accessible to the user).
  • Schedule tasks submitted by multiple users on one or more computers.

*in HTCondor-speak, a “pool”
User-Focused Tutorial

• For the purposes of this tutorial, we are assuming that someone else has set up HTCondor on a computer/computers to create a HTCondor “pool”.

• The focus of this talk is how to run computational work on this system.
Set Up and Run a Job with HTCondor
Jobs

• A single computing task is called a “job”
• Three main pieces of a job are the input, executable (program) and output

• Executable must be runnable from the command line without any interactive input
For our example, we will be using an imaginary program called “compare_states”, which compares two data files and produces a single output file.

```
$ compare_states wi.dat us.dat wi.dat.out
```
Job Translation

• Submit file: communicates everything about your job(s) to HTCondor

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB

queue 1
```
File Transfer

• Our example will use HTCondor’s file transfer option:

Access Point

(submit_dir)/
input files
executable

→

Execute Point

(execute_dir)/
transferred_files
output_files
job.submit

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

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

```bash
job.submit

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB
queue 1
```

- List your executable and any arguments it takes.

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

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

```bash
job.submit

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB
queue 1
```

- Indicate your input files.
  - `wi.dat`
  - `us.dat`

- HTCondor will transfer back all new and changed files (usually output) from the job.
  - `wi.dat.out`
Submit File

```plaintext
job.submit

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB
queue 1
```

- **log**: file created by HTCondor to track job progress

- **output/error**: captures stdout and stderr
Submit File

```plaintext
job.submit

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB
queue 1
```

- Request the appropriate resources for your job to run.

- `queue`: keyword indicating “create a job.”
Submitting and Monitoring Jobs

• To submit a job/jobs:
  `condor_submit submit_file_name`

• To monitor submitted jobs, use:
  `condor_q`

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

$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618>... @ 05/01/19 10:35:54
OWNER  BATCH_NAME  SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice  ID: 128  5/9  11:09  _  _  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:
  - user’s job(s) only, summarized in “batches”
- Constrain with username, `ClusterId` or full `JobId`, which will be denoted `[U/C/J]` in the following slides.

```
$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/09/19 11:35:54
OWNER  BATCH_NAME  SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice  ID: 128  5/9  11:09 _ _ 1  1 128.0
1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

`JobId = ClusterId.ProcId`
More about condor_q

• To see individual job information, use:

  `condor_q -nobatch`

```
$ condor_q -nobatch
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618>...
 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 wi.dat us.dat

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-1.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>I</td>
<td>0</td>
<td>compare_states wi.dat us.dat</td>
</tr>
</tbody>
</table>

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

Access Point

(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log
Job Starts

```bash
$ condor_q -nobatch
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618>?...
ID   OWNER  SUBMITTED  RUN_TIME  PRI SIZE  CMD
128.0 alice  5/9  11:09  0+00:00:00 < 0   0.0  compare_states wi.dat us.dat w

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

Access Point

```plaintext
(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log
```

Execute Point

```plaintext
(execute_dir)/

compare_states
wi.dat
us.dat
```

5/23/22
$ condor_q -nobatch

-- Schedd: submit-1.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>
<tbody>
<tr>
<td>128.0</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0+00:01:08</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

Access Point

(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log

Execute Point

(execute_dir)/

compare_states
wi.dat
us.dat
stderr
stdout
wi.dat.out
$ condor_q -nobatch
-- Schedd: submit-1.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</td>
<td>alice</td>
<td>5/9 11:09</td>
<td>0+00:02:02</td>
<td>&gt; 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

Access Point

(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log

Execute Point

(execute_dir)/

compare_states
wi.dat
us.dat
stderr
stdout
wi.dat.out

wi.dat.out

Job Completes (cont.)

$ condor_q -nobatch
-- Schedd: submit-1.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>
<tbody>
<tr>
<td>0</td>
<td>jobs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Access Point

(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log
job.out job.err
wi.dat.out
000 (7195807.000.000) 05/19 14:30:18 Job submitted from host: <128.105.244.191:9618 ...

......

040 (7195807.000.000) 05/19 14:31:55 Started transferring input files
   Transferring to host: <128.105.245.85:9618 ...

...

040 (7195807.000.000) 05/19 14:31:55 Finished transferring input files
...

001 (7195807.000.000) 05/19 14:31:56 Job executing on host: <128.105.245.85:9618 ...

...

005 (7195807.000.000) 05/19 14:35:56 Job terminated.
   (1) Normal termination (return value 0)

...

Partitionable Resources:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Usage</th>
<th>Request</th>
<th>Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Disk (KB)</td>
<td>26</td>
<td>1024</td>
<td>995252</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>1</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>
Job States

- **Idle (I)**: Transfer executable and input to execute point.
- **Running (R)**: Transfer output back to access point.
- **Completed (C)**: Job leaves the queue.

```
condorSubmit -> Idle (I) -> Running (R) -> Completed (C)
```
Assumptions

• Aspects of your submit file may be dictated by infrastructure and configuration.
  • For example: file transfer
    • previous example assumed files would need to be transferred between access point/execute point
      ```
      should_transfer_files = YES
      ```
    • not the case with a shared filesystem
      ```
      should_transfer_files = NO
      ```
Shared Filesystem

• If a system has a shared filesystem, where file transfer is not enabled, the submit directory and execute directory are the same.

```
shared_dir/
input
executable
output
```
job.submit

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

should_transfer_files = NO

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

request_cpus = 1
request_memory = 20MB
request_disk = 20MB

queue 1
```
Resource Request

• Jobs are nearly always using a part of a computer (a slot), not the whole thing.
• Very important to request appropriate resources (memory, cpus, disk) for a job.
Resource Assumptions

• Even if your system has default CPU, memory and disk requests, these may be too small! (or change, unexpectedly)

• Important to run test jobs and use the log file to request the right amount of resources:
  • requesting too little: causes problems for your and other jobs; jobs might by held by HTCondor
  • requesting too much: jobs will match to fewer “slots”, take longer to match
Submit Multiple Jobs with HTCondor
Why do we care?

• Run many independent jobs...
  • analyze multiple data files
  • test parameter or input combinations
  • and more!
Why do we care?

• Run many independent jobs...
  • analyze multiple data files
  • test parameter or input combinations
  • and more!
• ...without having to:
  • start each job individually
  • create separate submit files for each job
Many Jobs, One Submit File

- HTCondor has built-in ways to submit multiple independent jobs with one submit file.

See Rachel Lombardi’s talk next: Organizing and Submitting HTC Workloads

Photo by Joanna Kosinska on Unsplash
Numbered Input Files

• Goal: create 3 jobs that each analyze a different input file.

```bash
job.submit

executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in

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

queue
```

```bash
(submit_dir)/

analyze.exe
file0.in
gfile1.in
file2.in

job.submit
```
Multiple Jobs, No Variation

- This file generates 3 jobs, but doesn’t use multiple inputs and will overwrite outputs

```bash
job.submit

executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in

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

queue 3
```

```
(submit_dir)/

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

job.submit
```
Automatic Variables

• Each job’s ClusterId and ProcId can be accessed inside the submit file using:

$ \text{(ClusterId)}$

$ \text{(ProcId)}$

<table>
<thead>
<tr>
<th>ClusterId</th>
<th>ProcId</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>128</td>
<td>N-1</td>
</tr>
</tbody>
</table>
Job Variation

- How to uniquely identify each job (filenames, log/out/err names)?

```bash
job.submit

executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in
log = job.log
output = job.out
error = job.err
queue 3

(submit_dir)/

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

job.submit
```
Using $(ProcId)$

• Use the $(ClusterId)$, $(ProcId)$ variables to provide unique values to jobs.*

```plaintext
job.submit

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

log = job-$(ClusterId)-$(ProcId).log
output = job-$(ClusterId)-$(ProcId).out
error = job-$(ClusterId)-$(ProcId).err

queue 3
```

* May also see $(Cluster)$, $(Process)$ in documentation
Submit and Monitor (review)

```
condor_submit submit_file_name

condor_q
```

- Jobs in the queue will be grouped in batches (in this case by cluster number)

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

$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/09/19 10:35:54
OWNER BATCH_NAME SUBMITTED DONE RUN IDLE TOTAL JOB_IDS
alice ID: 128 5/9 11:03 _ _ 3 3 128.0-2

3 jobs; 0 completed, 0 removed, 3 idle, 0 running, 0 held, 0 suspended
```
Using Batches

- Alternatively, batches can be grouped manually using the `JobBatchName` attribute in a submit file:

```
JobBatchName = "CoolJobs"
```

```
$ condor_q
OWNER  BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CoolJobs      5/9  11:03     _     _     3     3 128.0–2
```

- To see individual jobs, use:

```
condor_q -nobatch
```
Organizing Jobs
Shared Files

• HTCondor can transfer an entire directory or all the contents of a directory
  • transfer whole directory
    \[
    \text{transfer\_input\_files} = \text{shared}
    \]
  • transfer contents only
    \[
    \text{transfer\_input\_files} = \text{shared/}
    \]

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

\[
\text{(submit\_dir)/}
\]

\[
\text{job.submit}
\]
\[
\text{shared/}
\]
\[
\text{reference.db}
\]
\[
\text{parse.py}
\]
\[
\text{analyze.py}
\]
\[
\text{cleanup.py}
\]
\[
\text{links.config}
\]
Shared Files

- HTCondor can transfer an entire directory or all the contents of a directory
  - transfer whole directory
    \[\text{transfer\_input\_files = shared}\]
  - transfer contents only
    \[\text{transfer\_input\_files = shared/}\]
- Useful for jobs with many shared files; transfer a directory of files instead of listing files individually

\[\text{(submit\_dir)/}\]

\[\text{job.submit}\]
\[\text{shared/}\]
\[\text{reference.db}\]
\[\text{parse.py}\]
\[\text{analyze.py}\]
\[\text{cleanup.py}\]
\[\text{links.config}\]
Use Sub-Directories for File Type

- Create sub-directories* and use paths in the submit file to separate input, error, log, and output files.

```plaintext
job.submit

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

log = log/job$(ProcId).log

queue 3
```

* must be created before the job is submitted
One Job per Directory

- Change the submission directory for each job using `initialdir`
- Allows the user to organize job files into separate directories.
- Use the same name for all input/output files
- Useful for jobs with lots of output files

job0  job1  job2  job3  job4
Separate Jobs with InitialDir

```
Executable should be in the directory with the submit file, *not* in the individual job directories
```
Output Handling

• Only transfer back specific files or directories from the job’s execution using `transfer_output_files`

• rename with `transfer_output_remaps`

```plaintext
transfer_output_files = results-final.dat, logs
transfer_output_remaps = "results-final.dat=results.dat"
```

```
<table>
<thead>
<tr>
<th>(submit_dir)/</th>
<th>(execute_dir)/</th>
</tr>
</thead>
<tbody>
<tr>
<td>results.dat</td>
<td>condor_exec.exe</td>
</tr>
<tr>
<td>logs/</td>
<td>results-tmp-01.dat</td>
</tr>
<tr>
<td></td>
<td>results-tmp-02.dat</td>
</tr>
<tr>
<td></td>
<td>results-tmp-03.dat</td>
</tr>
<tr>
<td></td>
<td>results-final.dat</td>
</tr>
<tr>
<td></td>
<td>logs/</td>
</tr>
</tbody>
</table>
```
Other Submission Methods

- What if your input files/directories aren’t numbered from 0 to \((N-1)\)?
- There are other ways to submit many jobs!
Submitting Multiple Jobs

Replacing single job inputs…

…with a variable of choice.

```bash
executable = compare_states
arguments = wi.dat us.dat wi.dat.out
transfer_input_files = us.dat, wi.dat
queue 1
```

```bash
executable = compare_states
arguments = $(infile) us.dat $(infile).out
transfer_input_files = us.dat, $(infile)
queue ...
```
Possible Queue Statements

<table>
<thead>
<tr>
<th>matching ... pattern</th>
<th>queue infile matching *.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ... list</td>
<td>queue infile in (wi.dat ca.dat ia.dat)</td>
</tr>
<tr>
<td>from ... file</td>
<td>queue infile from state_list.txt</td>
</tr>
</tbody>
</table>
Queue Statement Comparison

| matching .. pattern | Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories
|                    | Requires good naming conventions, |
| in .. list         | Supports multiple variables, all information contained in a single file, reproducible
|                    | Harder to automate submit file creation |
| from .. file       | Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible
|                    | Additional file needed |
Using Multiple Variables

- The "from" syntax supports using multiple variables from a list.

```plaintext
job.submit

executable = compare_states
arguments = -y $(option) -i $(file)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = $(file)

queue file,option from job_list.txt
```

```plaintext
job_list.txt

wi.dat, 2010
wi.dat, 2015
cn.dat, 2010
cn.dat, 2015
ia.dat, 2010
ia.dat, 2015
```
Other Features

• Match existing files or directories:

```
queue input matching files *.dat
queue directory matching dirs job*
```

• Submit multiple jobs with same input data

```
queue 10 input matching files *.dat
```

• Use other automatic variables: $(Step)

```
arguments = -i $(input) -rep $(Step)
queue 10 input matching files *.dat
```
(60 second) Pause

Questions so far?
Job Matching and Class Ad Attributes
The Central Manager

- HTCondor matches jobs with computers via a “central manager”.

access point

central manager

execute

execute
Class Ads

- HTCondor stores a list of information about each job and each computer.
- This information is stored as a “Class Ad”
- Class Ads have the format:
  - AttributeName = value
  - can be a boolean, number, or string

HTCondor Manual: Appendix A: Class Ad Attributes
Job Class Ad

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

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB
queue 1
```

+ HTCondor configuration*

```
RequestCpus = 1
Err = "job.err"
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd = "/home/alice/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week"
RequestDisk = 20480
NumJobStarts = 0
TransferInput = "us.dat,wi.dat"
Out = "job.out"
UserLog = "/home/alice/job.log"
RequestMemory = 20
...
```
Computer “Machine” Class Ad

HasFileTransfer = true
DynamicSlot = true
TotalSlotDisk = 4300218.0
TargetType = "Job"
TotalSlotMemory = 2048
Mips = 17902
Memory = 2048
UtsnameSysname = "Linux"
MAX_PREEMPT = ( 3600 * 72 )
OpSysMajorVer = 6
TotalMemory = 9889
OpSysName = "SL"
HasDocker = true
...

HTCondor configuration

=>

+ HTCondor configuration

5/23/22
Job Matching

• On a regular basis, the central manager reviews Job and Machine Class Ads and matches jobs to computers.
Job Execution

• (Then the access and execute points communicate directly.)
Class Ads for People

• Class Ads also provide lots of useful information about jobs and computers to HTCondor users and administrators

See later talk: What Are My Jobs Doing?

Photo by Roman Kraft on Unsplash
Finding Job Attributes

- Use the “long” option for `condor_q`
  ```bash
  condor_q -l JobId
  ```

  ```bash
  $ condor_q -l 128.0
  WhenToTransferOutput = "ON_EXIT"
  TargetType = "Machine"
  Cmd = "/home/alice/tests/htcondor_week/compare_states"
  JobUniverse = 5
  Iwd = "/home/alice/tests/htcondor_week"
  RequestDisk = 20480
  NumJobStarts = 0
  OnExitRemove = true
  TransferInput = "us.dat,wi.dat"
  UserLog = "/home/alice/tests/htcondor_week/job.log"
  RequestMemory = 20
  ...
  ```
Displaying Job Attributes

- Use the “auto-format” option:

```
condor_q [U/C/J] -af Attribute1 Attribute2 ...
```

```
$ condor_q -af ClusterId ProcId RemoteHost MemoryUsage

1725 116 slot1_1@e092.chtc.wisc.edu 1709
1725 118 slot1_2@e093.chtc.wisc.edu 1709
1725 137 slot1_8@e125.chtc.wisc.edu 1709
1725 139 slot1_7@e121.chtc.wisc.edu 1709
1861 0 slot1_5@c025.chtc.wisc.edu 196
1863 0 slot1_3@atlas10.chtc.wisc.edu 269
1864 0 slot1_25@e348.chtc.wisc.edu 245
1865 0 slot1_23@e305.chtc.wisc.edu 196
```
Selecting Job Attributes

• Use the "constraint" option, along with an expression for what jobs you want to look at:

\texttt{condor\_q \ [U/C/J] \ -constraint \ 'Attribute \ >/</== \ value'}

\begin{verbatim}
$ condor_q -constraint 'JobBatchName == "CoolJobs"'

<table>
<thead>
<tr>
<th>OWNER</th>
<th>BATCH_NAME</th>
<th>SUBMITTED</th>
<th>DONE</th>
<th>RUN</th>
<th>IDLE</th>
<th>TOTAL JOB_IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>CoolJobs</td>
<td>5/9 11:03</td>
<td>_</td>
<td>_</td>
<td>3</td>
<td>3 128.0-2</td>
</tr>
</tbody>
</table>
\end{verbatim}
Other Displays

• See the whole queue (all users, all jobs)

`condor_q -all`

```
$ condor_q -all

-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?...

<table>
<thead>
<tr>
<th>OWNER</th>
<th>BATCH_NAME</th>
<th>SUBMITTED</th>
<th>DONE</th>
<th>RUN</th>
<th>IDLE</th>
<th>HOLD</th>
<th>TOTAL JOB_IDS</th>
</tr>
</thead>
</table>
| alice | DAG: 128   | 5/9 02:52 | 982   | 2   | _    | _    | 1000 18888976.0 ...
| bob   | DAG: 139   | 5/9 09:21 | _     | 1   | 89   | _    | 180 18910071.0 ...
| alice | DAG: 219   | 5/9 10:31 | 1     | 997 | 2    | _    | 1000 18911030.0 ...
| bob   | DAG: 226   | 5/9 10:51 | 10    | _   | 1    | _    | 44 18913051.0    
| bob   | CMD: ce.sh | 5/9 10:55 | _     | _   | _    | 2    | 18913029.0 ...
| alice | CMD: sb    | 5/9 10:57 | _     | 2   | 998  | _    | 18913030.0-999  |
```
Class Ads for Computers

- as `condor_q` is to jobs, `condor_status` is to computers (or “machines”)

<table>
<thead>
<tr>
<th>Name</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>Activity</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>Activity</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:slot1@c001.chtc.wisc.edu">slot1@c001.chtc.wisc.edu</a></td>
<td>Unclaimed Idle</td>
<td>0.000</td>
<td>673</td>
<td>25+01</td>
<td>LINUX</td>
<td>X86_64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_1@c001.chtc.wisc.edu">slot1_1@c001.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+01</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_2@c001.chtc.wisc.edu">slot1_2@c001.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+00</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_3@c001.chtc.wisc.edu">slot1_3@c001.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+14</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_4@c001.chtc.wisc.edu">slot1_4@c001.chtc.wisc.edu</a></td>
<td>Unclaimed Idle</td>
<td>1.000</td>
<td>2693</td>
<td>19+19</td>
<td>LINUX</td>
<td>X86_64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_1@c002.chtc.wisc.edu">slot1_1@c002.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+04</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_2@c002.chtc.wisc.edu">slot1_2@c002.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+01</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1@c004.chtc.wisc.edu">slot1@c004.chtc.wisc.edu</a></td>
<td>Unclaimed Idle</td>
<td>0.010</td>
<td>645</td>
<td>25+05</td>
<td>LINUX</td>
<td>X86_64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot1_1@c004.chtc.wisc.edu">slot1_1@c004.chtc.wisc.edu</a></td>
<td>Claimed</td>
<td>1.000</td>
<td>2048</td>
<td>0+01</td>
<td>LINUX</td>
<td>X86_64</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Owner Claimed Unclaimed Matched Preempting Backfill Drain</th>
</tr>
</thead>
<tbody>
<tr>
<td>X86_64/LINUX 10962 0 10340 613 0 0 0 9</td>
</tr>
<tr>
<td>X86_64/WINDOWS 2 2 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Total 10964 2 10340 613 0 0 0 9</td>
</tr>
</tbody>
</table>

HTCondor Manual: condor_status
Find Machine Attributes

• Use same options as `condor_q`: to get attributes for a specific machine, use:

```
condor_status -l Slot/Machine
```

```bash
$ condor_status -l slot1_1@c001.chtc.wisc.edu
HasFileTransfer = true
COLLECTOR_HOST_STRING = "cm.chtc.wisc.edu"
TotalTimeClaimedBusy = 43334c001.chtc.wisc.edu
Mips = 17902
MAX_PREEMPT = ( 3600 * ( 72 - 68 * ( WantGlidein == true ) ) )
Requirements = ( START ) && ( IsValidCheckpointPlatform ) && ( WithinResourceLimits )
State = "Claimed"
OpSysMajorVer = 6
OpSysName = "SL"
...```
Useful Machine Attributes

• **Machine, Name**: name of the server, or slot
• **Cpus, Memory, Disk**: resources on that server
• **GPUs, GPUs_DeviceName**: number and type of GPUs
• **RemoteOwner**: Who is running
• **CPUModel**: type of CPU
• ...and more (see the manual)
Display Machine Attributes

• Use same options as `condor_q`, part 2, to display attributes, use

  `condor_status [Machine] -af Attribute1 Attribute2 ...`

$ condor_status e000.chtc.wisc.edu -af Name CPUs Memory Disk HasCHTCStaging

  slot1@e1013.chtc.wisc.edu 1 80116 82285091 false
  slot1_1@e1013.chtc.wisc.edu 1 768 12992383 false
  slot1_2@e1013.chtc.wisc.edu 2 1536 1332553 false
  slot1_3@e1013.chtc.wisc.edu 1 768 12992383 false
  slot1_4@e1013.chtc.wisc.edu 2 1536 1332553 false
  slot1_5@e1013.chtc.wisc.edu 2 1536 1332553 false
  slot1_7@e1013.chtc.wisc.edu 2 1536 1332553 false
  slot1_8@e1013.chtc.wisc.edu 1 2048 2331967 false
  slot1_9@e1013.chtc.wisc.edu 2 1536 1332553 false
Machine Attributes

- To summarize, use the "-compact" option

```
$ condor_status -compact
```

<table>
<thead>
<tr>
<th>Machine</th>
<th>Platform</th>
<th>Slots</th>
<th>Cpus</th>
<th>Gpus</th>
<th>TotalGb</th>
<th>FreeCpu</th>
<th>FreeGb</th>
<th>CpuLoad</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>e007.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>8</td>
<td>8</td>
<td></td>
<td>23.46</td>
<td>0</td>
<td>0.00</td>
<td>1.24</td>
<td>Cb</td>
</tr>
<tr>
<td>e008.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>8</td>
<td>8</td>
<td></td>
<td>23.46</td>
<td>0</td>
<td>0.46</td>
<td>0.97</td>
<td>Cb</td>
</tr>
<tr>
<td>e009.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>11</td>
<td>16</td>
<td></td>
<td>23.46</td>
<td>5</td>
<td>0.00</td>
<td>0.81 **</td>
<td></td>
</tr>
<tr>
<td>e010.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>8</td>
<td>8</td>
<td></td>
<td>23.46</td>
<td>0</td>
<td>4.46</td>
<td>0.76</td>
<td>Cb</td>
</tr>
<tr>
<td>matlab-build-1.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>1</td>
<td>12</td>
<td></td>
<td>23.45</td>
<td>11</td>
<td>13.45</td>
<td>0.00 **</td>
<td></td>
</tr>
<tr>
<td>matlab-build-5.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>0</td>
<td>24</td>
<td></td>
<td>23.45</td>
<td>24</td>
<td>23.45</td>
<td>0.04</td>
<td>Ui</td>
</tr>
<tr>
<td>mem1.chtc.wisc.edu</td>
<td>x64/SL6</td>
<td>24</td>
<td>80</td>
<td></td>
<td>1009.67</td>
<td>8</td>
<td>0.17</td>
<td>0.60 **</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Owner Claimed Unclaimed Matched Preempting Backfill Drain</th>
</tr>
</thead>
<tbody>
<tr>
<td>x64/SL6 10416             0 9984 427 0 0 0 0 5</td>
</tr>
<tr>
<td>x64/WinVista 2             2 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Total 10418                2 9984 427 0 0 0 0 5</td>
</tr>
</tbody>
</table>
Testing and Troubleshooting
What Can Go Wrong?

• Jobs can go wrong “internally”:
  • something happens after the executable begins to run

• Jobs can go wrong from HTCondor’s perspective:
  • A job can’t be started at all,
  • Uses too much memory,
  • Has a badly formatted executable,
  • And more...
“Live” Troubleshooting

• To log in to a job where it is running, use:

\texttt{condor\_ssh\_to\_job JobId}

```
$ condor_ssh_to_job 128.0
Welcome to slot1_31@e395.chtc.wisc.edu!
Your condor job is running with pid(s) 3954839.
```
Reviewing Failed Jobs

- A job’s log, output and error files can provide valuable information for troubleshooting

<table>
<thead>
<tr>
<th>Log</th>
<th>Output</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>• When jobs were submitted, started, and stopped</td>
<td>Any “print” or “display” information from your program</td>
<td>Captured by the operating system</td>
</tr>
<tr>
<td>• Resources used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exit status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Where job ran</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interruption reasons</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reviewing Recent Jobs

• To review a large group of jobs at once, use `condor_history [U/C/J]`
• 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 alice 5/11 09:52 0+00:07:37 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.1002 alice 5/11 09:52 0+00:08:03 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.1081 alice 5/11 09:52 0+00:03:16 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.944 alice 5/11 09:52 0+00:11:15 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.659 alice 5/11 09:52 0+00:26:56 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.1003 alice 5/11 09:52 0+00:07:38 C 5/11 15:59 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.962 alice 5/11 09:52 0+00:09:36 C 5/11 15:59 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.898 alice 5/11 09:52 0+00:13:47 C 5/11 15:59 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Held Jobs

• HTCondor will put your job on hold if there’s something YOU need to fix.

• A job that goes on hold is interrupted (all progress is lost) and kept from running again, but remains in the queue in the “H” state.

$ condor_q -nobatch
ID   OWNER  SUBMITTED   RUN_TIME   ST PRI SIZE CMD
128.0 alice 5/9 11:09 0+00:00:00 H 0 0.0 analyze.exe

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

• If HTCondor puts jobs on hold, it provides a hold reason, which can be viewed with:

  \[ \texttt{condor_q \ -\hold} \]

```bash
$ condor_q -hold
ID    OWNER    HELD_SINCE  HOLD_REASON
125.0 bob      5/09 17:12  Error from slot1_1@wid-003.chtc.wisc.edu: Job has gone over memory limit of 2048 megabytes.
128.0 alice    5/11 12:06  Error from slot1_11@e138.chtc.wisc.edu: 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
131.0 bob      5/12 09:02  Error from slot1_38@e270.chtc.wisc.edu: Failed to execute '/var/lib/condor/execute/slot1/dir_2471876/condor_exec.exe' with arguments 2: (errno=2: 'No such file or directory')
```
Fixing Holds

• Job attributes can be edited while jobs are in the queue using:

```
condor_qedit [U/C/J] Attribute Value
```

```
$ condor_qedit 128.0 RequestMemory 3072
Set attribute "RequestMemory".
```

• If a job has been fixed and can run again, release it with:

```
condor_release [U/C/J]
```

```
$ condor_release 128.0
Job 18933774.0 released
```
Holding or Removing Jobs

• If you know your job has a problem and it hasn’t yet completed, you can:
  • Place it on hold yourself, with `condor_hold [U/C/J]`

```bash
$ condor_hold bob
All jobs of user "bob" have been held

$ condor_hold 128
All jobs in cluster 128 have been held

$ condor_hold 128.0
Job 128.0 held
```

• Remove it from the queue, using `condor_rm [U/C/J]`

HTCondor Manual: condor_hold
HTCondor Manual: condor_rm
Job States, Revisited

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

in the queue

leaving the queue
Job States, Revisited

condor_submit

Idle (I)

condor_hold, or HTCondor puts a job on hold

condor_release

Running (R)

Completed (C)

in the queue

leaving the queue
Job States, Revisited

- **Idle** (I)
- **Running** (R)
- **Completed** (C)
- **Held** (H)
- **Removed** (X)

*not comprehensive

**Commands:**
- `condor_submit`
- `condor_release`
- `condor_hold`, or HTCondor puts a job on hold
- `condor_rm`
- `condor_q`
- `condor_history`
Use Cases and HTCondor Features
Interactive Jobs

• An interactive job proceeds like a normal batch job, but opens a bash session into the job’s execution directory instead of running an executable.

```bash
condor_submit -i submit_file
```

$ condor_submit -i interactive.submit
Submitting job(s).
1 job(s) submitted to cluster 18980881.
Waiting for job to start...
Welcome to slot1_9@e184.chtc.wisc.edu!

• Useful for testing and troubleshooting
Self-Checkpointing

• By default, a job that is interrupted will start from the beginning if it is restarted.

• It is possible to implement self-checkpointing, which will allow a job to restart from a saved state if interrupted.

• Self-checkpointing is useful for:
  • very long jobs
  • running on opportunistic resources.
Self-Checkpointing How-To

• Edit executable:
  • Regularly exit with a non-zero exit code, after saving intermediate states to a checkpoint file
  • Always check for a checkpoint file when starting
• Add HTCondor options that transfer checkpoint files back to the Access Point and then restarts the executable:

```
checkpoint_exit_code = 85
transfer_checkpoint_files = check.point
```

See Todd Miller’s afternoon talk: Self-Checkpointing Jobs
Job Universes

- HTCondor has different “universes” for running specialized job types
  - HTCondor Manual: Choosing an HTCondor Universe
- Vanilla (default)
  - good for most software
  - HTCondor Manual: Vanilla Universe
  - Set in the submit file using:
    \[ \text{universe} = \text{vanilla} \]
Docker Universe

- Use docker universe to
  - Run jobs inside a Docker container
  - **HTCondor Manual: Docker Universe Applications**

```python
universe = docker
docker_image = ubuntu:trusty
# by default the docker image
# is pulled from DockerHub
(execute_dir)/
  compare_states
  wi.dat
  us.dat
  stderr
  stdout
  wi.dat.out
```
Multi-CPU and GPU Computing

• Jobs that use multiple cores on a single computer can be run in the vanilla universe (parallel universe not needed):

```
request_cpus = 16
```

• If there are computers with GPUs, request them with:

```
request_gpus = 1
```
Automation
Automation

- After job submission, HTCondor manages jobs based on its configuration.
- You can use options that will customize job management even further.
- These options can automate when jobs are started, stopped, and removed.
Retries

- **Problem**: a small number of jobs fail; if they run again, they complete successfully.
- **Solution**: If the job exits with an error, leave it in the queue to run again. This is done via the automatic option `max_retries`.

```plaintext
max_retries = 5
```
Limiting Jobs

• **Problem**: Submitting more than a few thousand jobs to the queue at once

• **Solution**: Use the `max_idle` option. This limits the number of jobs submitted at one time, but allows there to always be idle jobs ready to run.

```plaintext
max_idle = 1000
```
Useful Job Attributes for Automation

- **CurrentTime**: current time
- **EnteredCurrentStatus**: time of last status change
- **ExitCode**: the exit code from the job
- **HoldReasonCode**: number corresponding to a hold reason
- **NumJobStarts**: how many times the job has gone from idle to running
- **JobStatus**: number indicating idle, running, held, etc.

[HTCondor Manual: Appendix A: JobStatus and HoldReason Codes](#)
Automatically Hold Jobs

• **Problem:** Your job should run in 2 hours or less, but a few jobs “hang” randomly and run for days

• **Solution:** Put jobs on hold if they run for over 2 hours, using a `periodic_hold` statement

```plaintext
periodic_hold = (JobStatus == 2) && 
((CurrentTime - EnteredCurrentStatus) > (60 * 60 * 2))
```

- **job is running**
- **How long the job has been running, in seconds**
- **2 hours**
Automatically Release Jobs

• **Problem** (related to previous): A few jobs are being held for running long; they will complete if they run again.

• **Solution**: automatically release those held jobs with a periodic_release option, up to 3 times

```
periodic_release = (JobStatus == 5) &&
(HoldReasonCode == 3) && (NumJobStarts < 3)
```

- job is held
- job was put on hold by periodic_hold
- job has started running less than 3 times
Automatically Remove Jobs

- **Problem**: Jobs are repetitively failing
- **Solution**: Remove jobs from the queue using a `periodic_remove` statement

```python
periodic_remove = (NumJobStarts > 5)
```

job has started running more than 5 times
Workflows

• **Problem**: Want to submit jobs in a particular order, with dependencies between groups of jobs

• **Solution**: Write a DAG

• To learn about this, stay for the later talk, [DAGMan: HTCondor and Workflows](#) by Lauren Michael.
Final Questions?
Acknowledgements

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.