AN INTRODUCTION TO USING HTCondor

Christina Koch
May 21, 2018
Covered In This Tutorial

• What is HTCondor?
• Running a Job with HTCondor
• Submitting 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 a submit point)
- HTCondor schedules them to run on computers (execute points)
HTCondor on One Computer
HTCondor on Many Computers
Why HTCondor?

- HTCondor manages and runs work on your behalf
- Schedule tasks on a single computer to not overwhelm the computer
- 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.

Setting up an HTCondor pool will be covered in “Administering HTCondor”, by Greg Thain, at 1:15 today (May 21)
Running 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
Job Example

- 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
```
File Transfer

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

Submit

(submit_dir)/
input files
executable

 Execute

(execute_dir)/
output files
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
```
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
```
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.
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
```

- Indicate your input files.
Submit File

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

• HTCondor will transfer back all new and changed files (usually output) from the job.
Submit File

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

job.submit

```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
```

- Request the appropriate resources for your job to run.
- `queue`: keyword indicating “create a job.”
Submitting and Monitoring

• 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-5.chtc.wisc.edu : <128.104.101.92:9618>?... @ 05/01/17 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 only (as of 8.6)
  – jobs summarized in “batches” (as of 8.6)

• Constrain with `username`, `ClusterId` or full `JobId`, which will be denoted `[U/C/J]` in the following slides

```bash
$ condor_q
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618>?... @ 05/09/17 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-5.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

```shell
$ 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>
<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>0.0</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
```

Submit Node

```plaintext
(submit_dir)/
  job.submit
  compare_states
  wi.dat
  us.dat
  job.log
```
Job Starts

```
$ condor_q -nobatch
-- Schedd: submit-5.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 <  0    0.0 compare_states wi.dat us.dat w

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
```

Execute Node

```
(execute_dir)/
  compare_states
  wi.dat
  us.dat
```
Job Running

$ condor_q -nobatch

-- Schedd: submit-5.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+00:01:08 R 0 0.0 compare_states wi.dat us.dat

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

Execute Node

(execute_dir)/
  compare_states
  wi.dat
  us.dat
  stderr
  stdout
  wi.dat.out
Job Completes

$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618>...
ID   OWNER    SUBMITTED  RUN_TIME  ST  PRI SIZE  CMD
128   alice    5/9 11:09  0+00:02:02 >  0  0.0  compare_states  wi.dat  us.dat

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

Execute Node
(execute_dir)/
  compare_states
  wi.dat
  us.dat
  stderr
  stdout
  wi.dat.out
Job Completes (cont.)

```bash
$ 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>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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
```
Log File

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
   Cpus       : 1       1
   Disk (KB)  : 14 20480 17203728
   Memory (MB): 1       20 20
Job States

condor_submit

Idle (I)

transfer executable and input to execute node

Running (R)

Completed (C)

transfer output back to submit node

in the queue

leaving the queue
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 submit/execute
  
  \[
  \text{should-transfer-files} = \text{YES}
  \]
  - not the case with a shared filesystem

  \[
  \text{should-transfer-files} = \text{NO}
  \]
Shared Filesystem

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

Submit

Execute

shared_dir/

input

executable

output
Shared Filesystem

job.submit

```
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, 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!

• 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”
Submitting Multiple Jobs with HTCondor
Why do we care?

- Run many independent jobs...
  - analyze multiple data files
  - test parameter or input combinations
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
Numbered Input Files

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

```bash
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
```
Multiple Jobs, No Variation

This file generates 3 jobs, but doesn’t use multiple inputs and will overwrite outputs.
### Automatic Variables

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

```
$(ClusterId)
$(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

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

(submit_dir)/
analyze.exe
file0.in
file1.in
file2.in
job.submit

• How to uniquely identify each job (filenames, log/out/err names)?
Using $(ProcId)$

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

```bash
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
(job.submit_dir)/

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

job.submit
```

* 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-5.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/01/17 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

12181445_0.err  16058473_0.err  17381628_0.err  18159900_0.err  5175744_0.err  7266263_0.err
12181445_0.log  16058473_0.log  17381628_0.log  18159900_0.log  5175744_0.log  7266263_0.log
12181445_0.out  16058473_0.out  17381628_0.out  18159900_0.out  5175744_0.out  7266263_0.out
13609567_0.err  16060330_0.err  17381640_0.err  3446080_0.err  5176204_0.err  7266267_0.err
13609567_0.log  16060330_0.log  17381640_0.log  3446080_0.log  5176204_0.log  7266267_0.log
13609567_0.out  16060330_0.out  17381640_0.out  3446080_0.out  5176204_0.out  7266267_0.out
13612268_0.err  16254074_0.err  17381665_0.err  3446306_0.err  5295132_0.err  7937420_0.err
13612268_0.log  16254074_0.log  17381665_0.log  3446306_0.log  5295132_0.log  7937420_0.log
13612268_0.out  16254074_0.out  17381665_0.out  3446306_0.out  5295132_0.out  7937420_0.out
13630381_0.err  17134215_0.err  17381676_0.err  4347054_0.err  5318339_0.err  8779997_0.err
13630381_0.log  17134215_0.log  17381676_0.log  4347054_0.log  5318339_0.log  8779997_0.log
13630381_0.out  17134215_0.out  17381676_0.out  4347054_0.out  5318339_0.out  8779997_0.out
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
  \[
  \text{job.submit}
  \]
  \[
  \text{shared/}
  \]
  \[
  \begin{align*}
  \text{reference.db} \\
  \text{parse.py} \\
  \text{analyze.py} \\
  \text{cleanup.py} \\
  \text{links.config}
  \end{align*}
  \]
Organize Files in Sub-Directories

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

* must be created before the job is submitted
Use Paths for File Type

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

log = log/job$(ProcId).log
error = err/job$(ProcId).err

queue 3
```
InitialDir

- 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
Separate Jobs with InitialDir

(submit_dir)/

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

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

```bash
job.submit

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

log = job.log
error = job.err

queue 3
```
Other Submission Methods

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

Replacing single job inputs with a variable of choice

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

transfer_input_files = us.dat, wi.dat
queue 1
```

```plaintext
executable = compare_states
arguments = $(infile) us.dat $(infile).out

transfer_input_files = us.dat, $(infile)
queue ...
```
### Possible Queue Statements

| multiple "queue" statements | `infile = wi.dat
queue 1
infile = ca.dat
queue 1
infile = ia.dat
queue 1` |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>matching ... pattern</td>
<td><code>queue infile matching *.dat</code></td>
</tr>
<tr>
<td>in ... list</td>
<td><code>queue infile in (wi.dat ca.dat ia.dat)</code></td>
</tr>
<tr>
<td>from ... file</td>
<td><code>queue infile from state_list.txt</code> wi.dat ca.dat ia.dat state_list.txt`</td>
</tr>
</tbody>
</table>
### Possible Queue Statements

| multiple “queue” statements | `infile = wi.dat
queue 1
infile = ca.dat
queue 1
infile = ia.dat
queue 1` |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>matching ... pattern</td>
<td><code>queue infile matching *.dat</code></td>
</tr>
<tr>
<td>in ... list</td>
<td><code>queue infile in (wi.dat ca.dat ia.dat)</code></td>
</tr>
<tr>
<td>from ... file</td>
<td><code>queue infile from state_list.txt</code></td>
</tr>
</tbody>
</table>

`state_list.txt`

**Not Recommended**
# Queue Statement Comparison

<table>
<thead>
<tr>
<th>multiple queue statements</th>
<th>Not recommended. Can be useful when submitting job batches where a single (non-file/argument) characteristic is changing</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>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>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>
Using Multiple Variables

- Both the “from” and “in” syntax support 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
ca.dat, 2010
ca.dat, 2015
ia.dat, 2010
ia.dat, 2015
```

HTCondor Manual: submit file options
Other Features

• Match existing files or directories:

<table>
<thead>
<tr>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue input matching files *.dat</td>
</tr>
<tr>
<td>queue directory matching dirs job*</td>
</tr>
</tbody>
</table>

• Submit multiple jobs with same input data

<table>
<thead>
<tr>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue 10 input matching files *.dat</td>
</tr>
</tbody>
</table>

Use other automatic variables: $(Step)

<table>
<thead>
<tr>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>arguments = -i $(input) -rep $(Step)</td>
</tr>
<tr>
<td>queue 10 input matching files *.dat</td>
</tr>
</tbody>
</table>
(60 SECOND) PAUSE

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

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

![Diagram of HTCondor](image)
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
Job Class Ad

RequestCpus = 1
Err = "job.err"
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
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job"
Out = "job.out"
UserLog = "/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...

*Configuring HTCondor will be covered in “Administering HTCondor”, by Greg Thain, at 1:15 today (May 21)
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 )
Requirements = ( START ) &&
( IsValidCheckpointPlatform ) &&
( WithinResourceLimits )
OpSysMajorVer = 6
TotalMemory = 9889
HasGluster = true
OpSysName = "SL"
HasDocker = true

...
Job Matching

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

• (Then the submit 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
Finding Job Attributes

• Use the “long” option for `condor_q`
  
  `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
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job"
UserLog = "'/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...```
Useful Job Attributes

- **UserLog**: location of job log
- **Iwd**: Initial Working Directory (i.e. submission directory) on submit node
- **MemoryUsage**: maximum memory the job has used
- **RemoteHost**: where the job is running
- **ClusterId, ProcID, JobBatchName**
- **...and more (see the manual)**
Displaying Job Attributes

- Use the “auto-format” option:

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

```bash
$ 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
1871 0 slot1_6@e176.chtc.wisc.edu 220
```
Selecting Job Attributes

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

```
condor_q [U/C/J] -constraint ‘Attribute >/</== value’
```

```
$ condor_q -constraint ‘JobBatchName == “CoolJobs”’

OWNER BATCH_NAME SUBMITTED DONE RUN IDLE TOTAL JOB_IDS
alice CoolJobs 5/9 11:03 _ _ 3 3 128.0-2
```
Other Displays

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

  condor_q -all

$ condor_q -all

-- Schedd: submit-5.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</th>
<th>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”)

```
$ condor_status
Name            OpSys    Arch    State       Activity   LoadAv  Mem   Activity
slot1@c001.chtc.wisc.edu LINUX    X86_64 Unclaimed Idle  0.000    673 25+01
slot1_1@c001.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+01
slot1_2@c001.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+01
slot1_3@c001.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+00
slot1_4@c001.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+14
slot1_5@c001.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    1024 0+01
slot1@c002.chtc.wisc.edu LINUX    X86_64 Unclaimed Idle 1.000    2693 19+19
slot1_1@c002.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+04
slot1_2@c002.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+01
slot1_3@c002.chtc.wisc.edu LINUX    X86_64 Claimed Busy 0.990    2048 0+02
slot1@c004.chtc.wisc.edu LINUX    X86_64 Unclaimed Idle 0.010    645 25+05
slot1_1@c004.chtc.wisc.edu LINUX    X86_64 Claimed Busy 1.000    2048 0+01

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

Total 10964  2  10340  613  0  0  0  0  9
```

HTCondor Manual: condor_status
Machine Attributes

- Use same options as `condor_q`:
  
  ```
  condor_status -l Slot/Machine
  condor_status [Machine] -af Attribute1 Attribute2 ...
  ```

```bash
$ condor_status -l slot1_1@c001.chtc.wisc.edu
HasFileTransfer = true
COLLECTOR_HOST_STRING = "cm.chtc.wisc.edu"
TargetType = "Job"
TotalTimeClaimedBusy = 43334c001.chtc.wisc.edu
UtsnameNodename = ""
Mips = 17902
MAX_PREEMPT = ( 3600 * ( 72 - 68 * ( WantGlidein == true ) ) )
Requirements = ( START ) && ( IsValidCheckpointPlatform ) &&
( WithinResourceLimits )
State = "Claimed"
OpSysMajorVer = 6
OpSysName = "SL"
...
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>FreCpu</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</th>
<th>Owner</th>
<th>Claimed</th>
<th>Unclaimed</th>
<th>Matched</th>
<th>Preempting</th>
<th>Backfill</th>
<th>Drain</th>
</tr>
</thead>
<tbody>
<tr>
<td>x64/SL6</td>
<td>10416</td>
<td>0</td>
<td>9984</td>
<td>427</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x64/WinVista</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| Total                          | 10418          | 2       | 9984      | 427     | 0          | 0        | 0     | 5     |
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...
# 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>
</table>
| • When jobs were submitted, started, and stopped
  • Resources used
  • Exit status
  • Where job ran
  • Interruption reasons | Any “print” or “display” information from your program | Captured by the operating system |
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

```
$ 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>189.653 alice 5/11 09:52 0+00:27:07 C 5/11 16:00 /home/alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189.1040 alice 5/11 09:52 0+00:05:15 C 5/11 15:59 /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>189.961 alice 5/11 09:52 0+00:09:43 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>
```
“Live” Troubleshooting

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

```bash
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.
```
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.
Diagnosing Holds

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

```bash
condor_q -hold
```

```bash
$ condor_q -hold
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  11:05  Error from slot1_20@e098.chtc.wisc.edu: 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>
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
128.0 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')
```
Common Hold Reasons

- Job has used more memory than requested
- Incorrect path to files that need to be transferred
- Badly formatted bash scripts (have Windows instead of Unix line endings)
- Submit directory is over quota
- The admin has put your job on hold
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

HTCondor Manual: condor_qedit
HTCondor Manual: condor_release
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]`
    
    ```
    $ 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]`
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

Held (H)

condor_release

Running (R)

Completed (C)

in the queue

leaving the queue
Job States, Revisited*

*not comprehensive
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.

  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
Output Handling

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

```
transfer_output_files = results-final.dat, logs
```

```
(submit_dir)/

(execute_dir)/
  condor_exec.exe
  results-tmp-01.dat
  results-tmp-02.dat
  results-tmp-03.dat
  results-tmp-04.dat
  results-tmp-05.dat
  results-final.dat
  logs/
```
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, and being able to run on opportunistic resources.
Self-Checkpointing How-To

• Edit executable:
  – Save intermediate states to a checkpoint file
  – Always check for a checkpoint file when starting

• Add HTCondor option that a) saves all intermediate/output files from the interrupted job and b) transfers them to the job when HTCondor runs it again

```plaintext
when_to_transfer_output = ON_EXIT_OR_EVICT
```
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:
  ```
  universe = vanilla
  ```
Other Universes

- **Standard**
  - Built for code (C, fortran) that can be statically compiled with `condor_compile`
    - [HTCondor Manual: Standard Universe](#)

- **Java**
  - Built-in Java support
    - [HTCondor Manual: Java Applications](#)

- **Local**
  - Run jobs on the submit node
    - [HTCondor Manual: Local Universe](#)

- **VM**
  - Run jobs inside a virtual machine
    - [HTCondor Manual: Virtual Machine Applications](#)

- **Parallel**
  - Used for coordinating jobs across multiple servers (e.g. MPI code)
  - Not necessary for single server multi-core jobs
    - [HTCondor Manual: Parallel Applications](#)

*Learn more about parallel universe from Jason Patton at 5:10pm today (May 21)*
Other Universes (cont.)

• Docker
  
  – Run jobs inside a Docker container

  [HTCondor Manual: Docker Universe Applications]

```
universe = docker
docker_image = ubuntu:trusty
# by default the docker image
# is pulled from DockerHub
```

Execute Node

Docker Container

(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):

  \[
  \text{request_cpus} = 16
  \]

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

  \[
  \text{request_gpus} = 1
  \]
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`.

\[
\text{max_retries} = 5
\]
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

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

- 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 5 times

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

- job is held
- job was put on hold by periodic_hold
- job has started running less than 5 times
Automatically Remove Jobs

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

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

job has started running more than 5 times
Relevant Job Attributes

- **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.
**Workflows**

- **Problem:** Want to submit jobs in a particular order, with dependencies between groups of jobs
- **Solution:** Write a DAG
  - To learn about this, attend the next talk, [DAGMan: HTCondor and Workflows](#) by Lauren Michael at 10:45 today (May 21).
FINAL QUESTIONS?