AN INTRODUCTION TO USING HTCondor

Christina Koch
Covered In This Tutorial

- What is HTCondor?
- Running a Job with HTCondor
- How HTCondor Matches and Runs Jobs
  - pause for questions -
- Submitting Multiple Jobs with HTCondor
- 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)
Single Computer

- Submit
- Execute
- Execute
- Execute
- Execute
Multiple Computers

HTCondor

submit

execute

execute

execute
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 1:05 today (May 17)
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.

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

- Our example will use HTCondor's file transfer option:

  ![Diagram showing file transfer process]

  - (submit_dir)/input files
  - executable

  ![Diagram showing file transfer process]

  - (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_disk = 20MB
request_memory = 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_disk = 20MB
request_memory = 20MB

queue 1
```

- List your executable and any arguments it takes.

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

- Arguments are any options passed to the executable from the command line.
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_disk = 20MB
request_memory = 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_disk = 20MB
request_memory = 20MB

queue 1

• HTCondor will transfer back all new and changed files (usually output) from the job.
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_disk = 20MB
request_memory = 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_disk = 20MB
request_memory = 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>...
  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
```
By default `condor_q` shows user’s job only*

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

* as of version 8.5
Job Idle

$ condor_q

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

Submit Node

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

```
$ condor_q
-- 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
  job.out
  job.err
```

Execute Node

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

$ condor_q

-- 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+00:01:08</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
Job Completes

```
$ condor_q
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618>...
ID    OWNER  SUBMITTED    RUN_TIME    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
  job.out
  job.err
```

Execute Node

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

$ condor_q

-- 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 + configuration

• For example: file transfer
  – previous example assumed files would need to be transferred between submit/execute

    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.
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 be held by HTCondor
  – requesting too much: jobs will match to fewer “slots”
Job Matching and Class Ad Attributes
The Central Manager

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

submit

execute

execute

execute

central manager
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/compar_e_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.lo g"
RequestMemory = 20

+ HTCondor configuration*

*Configuring HTCondor will be covered in “Administering HTCondor”, by Greg Thain, at 1:05 today (May 17)
Computer “Machine”
Class Ad

HTCondor configuration

HasFileTransfer = true
DynamicSlot = true
TotalSlotDisk = 4300218.0
TargetType = "Job"
TotalSlotMemory = 2048
Mips = 17902
Memory = 2048
UtsnameSysname = "Linux"
MAX_PREEMPT = ( 3600 * ( 72 - 68 * ( WantGlidein == true ) ) )
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
  ```

```
$ 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
• **BatchName**: optional attribute to label job batches
• ...and more
Displaying Job Attributes

• Use the “auto-format” option:
  
  condor_q [U/C/J] -af Attribute1 Attribute2 ...

$ condor_q -af ClusterId ProcId RemoteHost MemoryUsage

17315225  116 slot1_1@e092.chtc.wisc.edu 1709
17315225  118 slot1_2@e093.chtc.wisc.edu 1709
17315225  137 slot1_8@e125.chtc.wisc.edu 1709
17315225  139 slot1_7@e121.chtc.wisc.edu 1709
18050961  0 slot1_5@c025.chtc.wisc.edu 196
18050963  0 slot1_3@atlas10.chtc.wisc.edu 269
18050964  0 slot1_25@e348.chtc.wisc.edu 245
18050965  0 slot1_23@e305.chtc.wisc.edu 196
18050971  0 slot1_6@e176.chtc.wisc.edu 220
Other Displays

• See the whole queue (all users, all jobs)
  
  \texttt{condor\_q \ -all}

$\texttt{condor\_q \ -all}$

$\texttt{-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...}$

<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>233.0</td>
<td>alice</td>
<td>5/3</td>
<td>10:25</td>
<td>2+09:01:27</td>
<td>R</td>
<td>0</td>
<td>3663</td>
</tr>
<tr>
<td>240.0</td>
<td>alice</td>
<td>5/3</td>
<td>10:35</td>
<td>2+08:52:12</td>
<td>R</td>
<td>0</td>
<td>3663</td>
</tr>
<tr>
<td>248.0</td>
<td>alice</td>
<td>5/3</td>
<td>13:17</td>
<td>2+08:18:00</td>
<td>R</td>
<td>0</td>
<td>3663</td>
</tr>
<tr>
<td>631.6</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>631.7</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>631.8</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>631.9</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>631.10</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>631.16</td>
<td>bob</td>
<td>5/4</td>
<td>11:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Other Displays (cont.)

• See the whole queue, grouped in batches

  condor_q -all -batch

$ condor_q -all -batch

-- 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_test.sh | 5/9 10:55 | _    | _   | _    | 2    | _     | 18913029.0...
| alice | CMD: sb    | 5/9 10:57 | 2    | 998 | _    | _    | _     | 18913030.0-999 |

• Batches can be grouped manually using the BatchName attribute in a submit file: +JobBatchName = “CoolJobs”
• Otherwise HTCondor groups jobs automatically
Class Ads for Computers

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

```bash
$ condor_status
```

```
Name     OpSys    Arch  State               Activity  LoadAv  Mem Actvty
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

Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
X86_64/LINUX 10962   0  10340   613    0     0     0    9
X86_64/WINDOWS  2    2       0     0     0     0    0

Total 10964   2  10340   613    0     0     0    9

```

HTCondor Manual: `condor_status`

HTCondor Week 2016
Machine Attributes

• Use same options as `condor_q`:

```bash
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_q -compact
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>Platform</th>
<th>Owner 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>
</tr>
<tr>
<td>x64/WinVista</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
(60 SECOND) PAUSE

Questions so far?
Submitting Multiple Jobs with HTCondor
Many Jobs, One Submit File

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

• 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
Multiple, 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.

```plaintext
job.submit

executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file.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 numbers are saved as job attributes.
- They can be accessed inside the submit file using:
  - `${ClusterId}`
  - `${ProcId}`
Job Variation

exec = 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

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

```plaintext
job.submit

executable = analyze.exe
arguments = file$(ProcId).in file$(ProcId).out
should_transfer_files = YES
transfer_input_files = file$(ProcId).in
when_to_transfer_output = ON_EXIT

log = job_$(ClusterId).log
output = job_$(ClusterId)_$(_ProcId).out
error = job_$(ClusterId)_$(_ProcId).err

queue 3
```

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

* May also see $(Cluster), $(Process) in documentation
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
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
15348741_0.err  17134280_0.err  17382621_0.err  5024440_0.err  6842935_0.err  8839492_0.err
15348741_0.log  17134280_0.log  17382621_0.log  5024440_0.log  6842935_0.log  8839492_0.log
15348741_0.out  17134280_0.out  17382621_0.out  5024440_0.out  6842935_0.out  8839492_0.out
15741282_0.err  17381597_0.err  17392160_0.err  5175145_0.err  6882517_0.err  8873254_0.err
15741282_0.log  17381597_0.log  17392160_0.log  5175145_0.log  6882517_0.log  8873254_0.log
15741282_0.out  17381597_0.out  17392160_0.out  5175145_0.out  6882517_0.out  8873254_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
    
    \[
    \text{job.submit} \\
    \text{shared/} \\
    \text{reference.db} \\
    \text{parse.py} \\
    \text{analyze.py} \\
    \text{cleanup.py} \\
    \text{links.config}
    \]
    
    \[(\text{submit\_dir})/\]
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

**Command:**
```bash
job.submit
```

**Path Configuration:**
```bash
 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
```

**Queueing:**
```bash
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

<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>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.

```plaintext
job.submit

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

log = job.log
table: 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

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

| Multiple “queue” statements | \begin{verbatim}
infile = wi.dat
queue 1
infore = ca.dat
queue 1
infore = ia.dat
queue 1
\end{verbatim} |
|-----------------------------|--------------------------------------------------|
| Matching ... pattern        | \begin{verbatim}
queue infile matching *.dat
\end{verbatim} |
| In ... list                 | \begin{verbatim}
queue infile in (wi.dat ca.dat ia.dat)
\end{verbatim} |
| From ... file               | \begin{verbatim}
queue infile from state_list.txt
\end{verbatim}

\begin{itemize}
\item wi.dat
\item ca.dat
\item ia.dat
\item state_list.txt
\end{itemize}
**Possible Queue Statements**

| multiple "queue" statements | infile = wi.dat  
|                             | queue 1  
|                             | infile = ca.dat  
|                             | queue 1  
|                             | infile = ia.dat  
|                             | queue 1  
| matching ... pattern | queue infile matching *.dat  
| in ... list | queue infile in (wi.dat ca.dat ia.dat)  
| from ... file | queue infile from state_list.txt  
|                | wi.dat  
|                | ca.dat  
|                | ia.dat  
|                | 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 &quot;files&quot; and &quot;dirs&quot; 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.

```bash
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 only 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

- Come to TJ’s talk: **Advanced Submit** at 4:25 today
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>
<tbody>
<tr>
<td>When jobs were submitted,</td>
<td>Any “print” or “display” information from</td>
<td>Ecaptured by the operating system</td>
</tr>
<tr>
<td>started, and stopped</td>
<td>your program</td>
<td></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 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

```
$ 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 C</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 C</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 C</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 C</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 C</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 C</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 C</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 C</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 C</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 C</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 C</td>
<td>C</td>
<td>5/11 15:59</td>
<td>/home/alice</td>
</tr>
</tbody>
</table>
```
“Live” Troubleshooting

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

  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 a job on hold, it provides a hold reason, which can be viewed with:

  `condor_q -hold`

```
$ condor_q -hold
128.0 alice  5/2 16:27 Error from slot1_1@wid-003.chtc.wisc.edu: Job has gone over memory limit of 2048 megabytes.
174.0 alice  5/5 20:53 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>
319.959 alice  5/10 05:23 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
534.2 alice  5/10 09:46 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:

\texttt{condor\_qedit \[U/C/J\] Attribute Value}

\begin{verbatim}
$ condor_qedit 128.0 RequestMemory 3072
Set attribute "RequestMemory".
\end{verbatim}

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

\texttt{condor\_release \[U/C/J\]}

\begin{verbatim}
$ condor_release 128.0
Job 18933774.0 released
\end{verbatim}
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

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.

```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
Output Handling

- Only transfer back specific files from the job’s execution using `transfer_output_files`

`transfer_output_files = results-final.dat`
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

`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
cstatically 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
Other Universes (cont.)

- Docker
  - Run jobs inside a Docker container
    HTCondor Manual: Docker Universe Applications

- 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
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 with a known error code; if they run again, they complete successfully.

• Solution: If the job exits with the error code, leave it in the queue to run again

```on_exit_remove = (ExitBySignal == False) && (ExitCode == 0)```
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 \texttt{periodic\_release} option, up to 5 times

\begin{verbatim}
periodic_release = (JobStatus == 5) &&
(HoldReason == 3) && (NumJobStarts < 5)
\end{verbatim}
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
Automatic Memory Increase

- Putting all these pieces together, the following lines will:
  - request a default amount of memory (2GB)
  - put the job on hold if it is exceeded
  - release the job with an increased memory request

request_memory = ifthenelse(MemoryUsage !== undefined, (MemoryUsage * 3/2), 2048)
periodic_hold = (MemoryUsage >= ((RequestMemory) * 5/4 )) && (JobStatus = 2)
periodic_release = (JobStatus == 5) && ((CurrentTime - EnteredCurrentStatus) > 180) && (NumJobStarts < 5) && (HoldReasonCode !== 13) && (HoldReasonCode !== 34)
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.
- **MemoryUsage**: how much memory the job has used
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 Kent Wenger at 10:45 today (May 17).
FINAL QUESTIONS?