HTCondor User Tutorial

CONTENTS

• Overview
  Basic job submission
  How submit files work
• Goals:
  Learn how to create submit files
What is (HT)Condor?

• Mature, Open Source HTC Implementation
  – Started in 1990, Current version 8.5.x
• Runs on Linux, Mac, Windows, etc.
• Part of RHEL, Debian, Fedora, etc.
• Reliable, Scalable, but Configurable
Two Fundamental Abstractions

- **Jobs**
  - Send job and input files to worker machine

- **Machines**
  - Send output files back to submitter
Two Fundamental User Types

Users create jobs

Admins create machines
This talk

• “User Talk”:
  – How to create/manage/optimize jobs

• But admins need to know this too
• Assume an admin has created a pool for us

• (Next talk: how to create/manage machines)
HTCondor fundamentals

• HTCondor is a distributed system
• Configured with text files
• Controlled by a Command Line Interface (CLI)
  – Most of this talk is about the using the CLI

• Not a demo!
• But can download/install from:
First steps

• Divide your work into self-describing jobs
  – This is an art/craft
• Create jobs
• Submit them to one or more queues
• Manage/Wait for output
Breaking your work into jobs

• Not entirely on topic here, considerations:
  – Job runtime
  – I/O requirements
  – Dependencies
  – Self-describing
Sample job

```
$ gzip input_file
$
```
Requirements?

$ gzip input_file

$
Requirements?

$ gzip input_file$

• Memory
• Scratch disk
• Executable
• Input file
• Output disk size

• # of CPUs/cores
• OS
• CPU Type
• Run time?
Submit Files describe jobs

• Submit files have many options
• Much of a user’s work is creating good ones
• Enormous number of options, few needed
• $ man condor_submit
• HTCondor manual
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
Example HTCondor submit file

```
Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
```

Universe means “job type”
Boiler plate for now
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = on_exit
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue

Path to program to start
Could be relative
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
Example HTCondor submit file

```
Universe = vanilla
Executable = /usr/bin/gzip
Arguments = inputs
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
```

Force HTCondor not to use shared filesystem
Example HTCondor submit file

```
Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
```

Comma separated
List of input files
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue

Force HTCondor not to use shared filesystem
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
Example HTCondor `submit file`

```
Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
```

Scratch disk to reserve
Example HTCondor submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = inputs
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue

# of cores to reserve
Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue

File for HTCondor to tell you about the life of job.
What’s Missing?

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = inputs
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue

Defaults:
Operating System
CPU Type
What to do with submit file?

$ condor_submit gzip.sub
Submitting job(s).
1 job(s) submitted to cluster 5.

• Must be run on “submit node”
If not a submit node?

$ condor_submit gzip.sub

ERROR: Can't find address of local schedd

• Not your fault – find an admin.
Submit is good, now what?

$ condor_submit gzip.sub
Submitting job(s).
1 job(s) submitted to cluster 5.

• Cluster ID returned important – primary key
condor_q: our 2nd command

$ condor_q 5
-- Schedd: s1.cs.wisc.edu : <127.0.0.1?...
ID    OWNER      SUBMITTED    RUN_TIME   ST PRI SIZE CMD
5.0   gthain     9/5 11:40    0+00:00:00 I  0   0.1 gzip
input
condor_q: our 2nd command

$ condor_q 5
-- Schedd: s1.cs.wisc.edu: <127.0.0.1?...

<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>5.0</td>
<td>gthain</td>
<td>9/5 11:40</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip inp</td>
</tr>
</tbody>
</table>

- Cluster ID: Note full name is “5.0”
condor_q: our 2nd command

$ condor_q 5
-- Schedd: s1.cs.wisc.edu : <127.0.0.1?...

<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>5.0</td>
<td>gthain</td>
<td>9/5 11:40</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip input</td>
</tr>
</tbody>
</table>

• Command with arguments
condor_q: our 2\textsuperscript{nd} command

$ condor_q 5

-- Schedd: s1.cs.wisc.edu : <127.0.0.1?...  

<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>5.0</td>
<td>gthain</td>
<td>9/5 11:40</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip input</td>
</tr>
</tbody>
</table>

- Job state: “I” = Idle
condor_q: our 2nd command

$ condor_q 5
-- Schedd: s1.cs.wisc.edu : <127.0.0.1?...

<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>5.0</td>
<td>gthain</td>
<td>9/5</td>
<td>11:40</td>
<td>0+00:00:04</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

• Job state: “R” = Running
condor_q: our 2\textsuperscript{nd} command

$ condor_q 5
-- Schedd: s1.cs.wisc.edu : <127.0.0.1?

<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>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>gthain</td>
<td>9/5 11:40</td>
<td>0+00:00:04</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td></td>
</tr>
</tbody>
</table>

• Job state: “R” = Running
condor_q: our 2\textsuperscript{nd} command

\$ condor_q 5
-- Schedd: s1.cs.wisc.edu : <127.0.0.1?...

<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 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• No output means job is completed.
Job states

- “I” Idle – no machine found
- “R” Running

- “I” Idle with runtime – has been evicted
- “H” Held – needs user attention
Log file contents

$ cat log
000 (005.000.000) 8/05 11:40:49 Job submitted from host: <127.0.0.1:51923?addrs=128.105.14.141-51923>
...
001 (005.000.000) 8/05 11:40:59 Job executing on host: <127.0.01:55459?addrs=128.105.14.141-55459>
...
005 (005.000.000) 8/05 11:41:00 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 - Total

RemoteUsage
32  - Run Bytes Sent By Job
68710  - Run Bytes Received By Job
32  - Total Bytes Sent By Job
68710  - Total Bytes Received By Job

Partitionable Resources : Usage Request Allocated
Cpus : 1 2
Disk (KB) : 76 10240 1041700
Memory (MB) : 10 100 649
...
One job is not HTC

• Easy example – parameter sweep in gzip
• Test gzip -# options and file sizes

$ gzip -1 input
$ gzip -2 input
$ gzip -3 input
...
$ gzip -9 input
Same submit file

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
Obvious solution

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue
A better way…

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = input
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log

Queue
Arguments = -1 input
Queue
Arguments = -2 input
Queue
An even better way

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = \(-\$(\text{PROCESS})\) input
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue 10

Log always appended
$ condor_q 8

<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>8.0</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -0 input</td>
</tr>
<tr>
<td>8.1</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -1 input</td>
</tr>
<tr>
<td>8.2</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -2 input</td>
</tr>
<tr>
<td>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -3 input</td>
</tr>
<tr>
<td>8.4</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -4 input</td>
</tr>
<tr>
<td>8.5</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -5 input</td>
</tr>
<tr>
<td>8.6</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -6 input</td>
</tr>
<tr>
<td>8.7</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -7 input</td>
</tr>
<tr>
<td>8.8</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -8 input</td>
</tr>
<tr>
<td>8.9</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -9 input</td>
</tr>
</tbody>
</table>
$ condor_q 8.3

<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>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -3 input</td>
</tr>
</tbody>
</table>

- 8.3 is job’s “full name” or job id
- 8 is cluster
- .3 is “process”
- Commands often take full name or cluster
Ooops, I just did “queue 100000”

$ condor_rm 8.3
job 8.3 marked for removal
$ condor_rm 8
all jobs in cluster 8 marked for removal

condor_rm removes jobs from queue
no matter what state
if running, kills running job and cleans up
Two surprises with example

$ condor_q 8

<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>8.0</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-0 input</td>
</tr>
<tr>
<td>8.1</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-1 input</td>
</tr>
<tr>
<td>8.2</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-2 input</td>
</tr>
<tr>
<td>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-3 input</td>
</tr>
<tr>
<td>8.4</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-4 input</td>
</tr>
<tr>
<td>8.5</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-5 input</td>
</tr>
<tr>
<td>8.6</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-6 input</td>
</tr>
<tr>
<td>8.7</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-7 input</td>
</tr>
<tr>
<td>8.8</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-8 input</td>
</tr>
<tr>
<td>8.9</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02 R</td>
<td>0</td>
<td>0.1</td>
<td>gzip</td>
<td>-9 input</td>
</tr>
</tbody>
</table>
Ooops:

$ gzip -0 input
gzip: invalid option -- '0'
Try `gzip --help' for more information.

$(PROCESS) is always zero-based
gzip doesn’t like that
Many workloads OK with zero basing
Fix for surprise #1

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = \$\{(Compression)\} input
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue 1 Compression in 1,2,3,4,5,6,7,8,9
Surprise # 2

$ gzip -1 input
(output is input.gz)

$ gzip -2 input
(output is input.gz)

Want to be able to compare sizes of input.gz
Each run overwrites the next.
Potential fix (doesn’t work)

Universe = vanilla
Executable = /usr/bin/gzip
Arguments = -$(C) input -o input.$(C).gz
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue 1 C in 1,2,3,4,5,6,7,8,9
Gzip doesn’t have -o

$ gzip input -o input.gz
gzip: invalid option -- 'o'
Try `gzip --help' for more information.
$ gzip -d input > input.gz

• Problem:
Problem: submit args just args

• Submit file arguments just arguments
• <, >, 2>&1, etc. don’t work there

• Job’s stdin, stdout, stderr listed separately:
  • Input = stdin_file
  • Output = stdout_file
  • Error = stderr_file
Universes = vanilla
Executable = /usr/bin/gzip
Arguments = -$(C) input -d
Input = input.$(C).gz
Transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue 1 C in 1,2,3,4,5,6,7,8,9

gzip.sub
Workflow Completely Solved

• Time to solve a harder more interesting issue

• But first, some interesting helpful tools…
Held state

• A Held job is unstartable. But still in queue
• User or System can hold job/cluster of jobs:

```
$ condor_hold -reason "bad data" 8.3
(or condor_hold -reason "bad data" gthain)
Job 8.3 held
$ condor_q -hold 8.3
ID   OWNER  HELD_SINCE HOLD_REASON
8.3 gthain  9/5  14:14 bad data (by gthain)
```
Getting out of held

- condor_release is opposite of condor_hold
- User can release job/cluster of jobs:

$ condor_release 8.3
(or condor_release gthain)
Job 8.3 release
$ condor_q 8.3
ID  OWNER  SUBMITTED  RUN_TIME  ST  PRI  SIZE  CMD
8.3  gthain  9/5  14:12  0+00:00:03  I  0  0.1  gzip -3 input
Remember: Jobs are atomic!

• `condor_release` RESTARTS jobs from start

• Any intermediate work is lost

• `WHEN_TO_TRANSFER_OUTPUT = \` 
  `ON_EXIT_OR_EVICT`
$ condor_q -hold 11
ID  OWNER  HELD_SINCE  HOLD_REASON
11.3 gthain 9/5 14:14 EMAIL lmichael NOW! (by root)
$ condor_q gthain
12.0 gthain 9/5 19:24 Error from slot1@foo: SHADOW at 128.105.14.141 failed to send file(s) to <128.105.14.141:40864>: error reading from /scratch/gthain/ptest/input.foo: (errno 2) No such file or directory; STARTER failed to receive file(s) from <128.105.14.141:57763>
condor_history

• When job finishes, goes to the history file
  – (for a limited time, depending on file size)
• condor_history reads historical jobs
  – (mostly completed and removed)
  – Same options as condor_q, mostly
## condor_history example

```
$ condor_history 8

<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>8.0</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -0 input</td>
</tr>
<tr>
<td>8.1</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -1 input</td>
</tr>
<tr>
<td>8.2</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -2 input</td>
</tr>
<tr>
<td>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -3 input</td>
</tr>
<tr>
<td>8.4</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -4 input</td>
</tr>
<tr>
<td>8.5</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -5 input</td>
</tr>
<tr>
<td>8.6</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -6 input</td>
</tr>
<tr>
<td>8.7</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -7 input</td>
</tr>
<tr>
<td>8.8</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -8 input</td>
</tr>
<tr>
<td>8.9</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>C</td>
<td>0</td>
<td>0.1</td>
<td>gzip -9 input</td>
</tr>
</tbody>
</table>
```
Job Priority

• When > 1 user
  – HTCondor decides how much each user get
• User choses which of THEIR job goes next

• Priority = Number
• Higher is better, range is integers
Job Priority Example

Universe = vanilla
Executable = /usr/bin/gzip
Arguments =
transfer_input_files = input
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log

**Priority = 100**
Queue 1
### Job Priority example

$ condor_q 8

<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>8.0</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -0 input</td>
</tr>
<tr>
<td>8.1</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -1 input</td>
</tr>
<tr>
<td>8.2</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -2 input</td>
</tr>
<tr>
<td>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:03</td>
<td>R</td>
<td>100</td>
<td>0.1</td>
<td>gzip -3 input</td>
</tr>
<tr>
<td>8.4</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -4 input</td>
</tr>
<tr>
<td>8.5</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -5 input</td>
</tr>
<tr>
<td>8.6</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -6 input</td>
</tr>
<tr>
<td>8.7</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -7 input</td>
</tr>
<tr>
<td>8.8</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -8 input</td>
</tr>
<tr>
<td>8.9</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.1</td>
<td>gzip -9 input</td>
</tr>
</tbody>
</table>
Job Environment

- By default, HTCondor will clear out Environment
- Can assign environment in submit file

```plaintext
Environment = “Env1=val1 Env2=val2”
```

- Or:

```plaintext
getenv = true
```

- To copy all environment vars from submitter
New Workload:

- Want to gzip many files.
- With a clean directory hierarchy:
What we want:

- `sdir/run.sub`
- `sdir/run.exe`
- `sdir/common.in`
- `sdir/0/`
- `sdir/0/input`
- `sdir/1/`
- `sdir/1/input`

- `sdir/0/output`
- `sdir/1/output`
Warning! Warning!

`transfer_input_files` takes relative paths:

```
transfer_input_files = foo, ../bar, /bin/ls
```

However, HTCondor squashes input files to a common scratch directory for execution:

```
$ ls
foo bar ls
```
InitialDir command can help

• All submit-side transfer prefixed with InitialDir

• InitialDir ignored on execute side
  – (except when using shared filesystem…)


What we want:

- `sdir/run.sub`
- `sdir/run.exe`
- `sdir/common.in`
- `sdir/0/`
- `sdir/0/input`
- `sdir/1/`
- `sdir/1/input`

- `sdir/run.sub`
- `sdir/run.exe`
- `sdir/common.in`
- `sdir/0/`
- `sdir/0/input`
- `sdir/0/output`
- `sdir/1/`
- `sdir/1/input`
- `sdir/1/output`
Submit file example

Universe = vanilla
Executable = run.exe
Arguments = common.in input
transfer_input_files = ../common.in, input
InitialDir = $(PROCESS)
Should_transfer_files = yes
When_to_transfer_output = on_exit
Request_memory = 100M
Request_disk = 1G
Request_cpus = 1
Log = log
Queue 10
DAGman and dependencies
Workflows

• Often, you don’t have independent tasks!

• Common example:
  – You want to analyze a set of images
    1. You need to generate N images (once)
    2. You need to analyze all N images
       One job per image
    3. You need to summarize all results (once)
Do you really want to do this manually?
Workflows: The HTC definition

Workflow:

A graph of jobs to run: one or more jobs must succeed before one or more others can start running
Example of a LIGO Inspiral DAG
DAGMan

• DAGMan: HTCondor’s workflow manager
  Directed Acyclic Graph (DAG) Manager (Man)

• Allows you to specify the dependencies between your HTCondor jobs

• Manages the jobs and their dependencies

• That is, it manages a workflow of HTCondor jobs
What is a DAG?

- A DAG is the structure used by DAGMan to represent these dependencies.
- Each job is in a node in the DAG.
- Each node can have any number of “parent” or “children” nodes – as long as there are no loops!
So, what’s in a node?

(optional pre-script)
Job
(optional post-script)
Defining a DAG

- A DAG is defined by a `.dag` file, listing each of its nodes and their dependencies. For example:

```plaintext
# Comments are good
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub

Parent A Child B C
Parent B C Child D
```
DAG Files…

- This complete DAG has five files

One DAG File:

- Job A a.sub
- Job B b.sub
- Job C c.sub
- Job D d.sub

Parent A Child B C
Parent B C Child D

Four Submit Files:

- Universe = Vanilla
- Executable = analysis...
- Universe = …
Submitting a DAG

- To start your DAG, just run `condor_submit_dag` with your .dag file, and HTCondor will start a DAGMan process to manage your jobs:

  ```bash
  % condor_submit_dag diamond.dag
  ```

- `condor_submit_dag` submits a Scheduler Universe job with DAGMan as the executable

- Thus the DAGMan daemon itself runs as an HTCondor job, so you don’t have to baby-sit it
DAGMan is a HTCondor job

- DAGMan itself is a condor job with a job id, so

  \% condor_rm job_id_of_dagman
  \% condor_hold job_id_of_dagman
  \% condor_q -dag # is magic

- DAGMan submits jobs, one cluster per node
- Don’t confuse dagman as job with jobs of dagman
Running a DAG

- DAGMan acts as a job scheduler, managing the submission of your jobs to HTCondor based on the DAG dependencies.
Running a DAG (cont’d)

- **DAGMan submits jobs to HTCondor at the appropriate times**
- **For example, after A finishes, it submits B & C**
Finishing a DAG

• Once the DAG is complete, the DAGMan job itself is finished, and exits
Successes and Failures

- A job *fails* if it exits with a non-zero exit code.
- In case of a job failure, DAGMan runs other jobs until it can no longer make progress, and then creates a "rescue" file with the current state of the DAG.
Recovering a DAG

• Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG

  – Another example of reliability for HTC!
Recovering a DAG (cont’d)

• Once that job completes, DAGMan will continue the DAG as if the failure never happened
DAGMan & Fancy Features

• DAGMan doesn’t have a lot of “fancy features”
  – No loops
  – Not much assistance in writing very large DAGs (script it yourself)
• Focus is on solid core
  – Add the features people need in order to run large DAGs well
  – People build systems on top of DAGMan
DAGMan: Reliability

- For each job, HTCondor generates a log file
- DAGMan reads this log to see what has happened
- If DAGMan dies (crash, power failure, etc…)
  - HTCondor will restart DAGMan
  - DAGMan re-reads log file
  - DAGMan knows everything it needs to know
  - Principle: DAGMan can recover state from files and without relying on a service (HTCondor queue, database…)
- Recall: HTC requires reliability!
**condor_ssh_to_job**

- Allows user to investigate running job
- E.g. if running too long, etc.

```
$ condor_q 8.3

<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>8.3</td>
<td>gthain</td>
<td>8/5 12:31</td>
<td>0+00:00:02</td>
<td>R</td>
<td>0</td>
<td>0.1</td>
<td>gzip -3 input</td>
</tr>
</tbody>
</table>

$ condor_ssh_to_job 8.3
Welcome to slot2@some_machine
Your condor job is running with pid(s) 228722.
[user@slot dir_xx] $```

Advanced File Transfer: subdirectories

Transfer_input_files = dir

Above means make dir on execute machine

Transfer_input_files = dir/

Above means collapse contents of dir in .
Advanced File Transfer: Plugins

Transfer_input_files = \http://example.com/dir/file
Above means xfer file from http server

Output_destination = http://...
Above mean deposit sandbox on serve
Admins can add more URLs as needed...
A word about wrappers

- Commonly used
- Need to transfer “real” executable
- Careful to check and return real exit status
ClassAds

the language that HTCondor uses to represent information about:
jobs \( (\text{job ClassAd}) \),
machines \( (\text{machine ClassAd}) \), and programs that implement HTCondor's functionality (called daemons)
Part of a Job ClassAd

MyType = "Job"
TargetType = "Machine"
ClusterId = 1
ProcID = 0
IsPhysics = True
Owner = "chris"
Cmd = "science.exe"
Requirements = (Arch == "INTEL")
Advanced Classad Usage

ON_EXIT_REMOVE = FALSE

ON_EXIT_REMOVE = ExitStatus == 0
Advanced Classad Usage

Requirements = Purchaser == "Chemistry"

PERIODIC_REMOVE = QDdate
Discover clasad attributes?

condor_q -l job_id

condor_status -l
Job Universes: Types of jobs

- Vanilla: “default”
- Standard: Checkpointing if you recompile
- Docker: ability to run containers as jobs
- Grid: submit to other schedulers
- VM: run a virtual machine image
- Scheduler: run on the submit machine
Thank you!

- Where to go for more info
  - Talk to us!
  - Condor Week 2016
  - gthain@cs.wisc.edu
  - http://htcondorproject.org
  - 800+ page Condor manual