An Introduction to Using HTCondor 2013
Established in 1985, to do research and development of distributed high-throughput computing
HTCondor does High-Throughput Computing

› Allows for many computational tasks to be completed over a long period of time
› Useful for researchers and other users who are more concerned with the number of computations they can do over long spans of time than they are with short-burst computations
HTCondor’s strengths

› Cycle scavenging works!
› High-throughput computing
› Very configurable, adaptable
› Supports strong security methods
› Interoperates with many types of computing grids
› Has features to manage both dedicated CPUs (clusters) and non-dedicated resources (desktops)
› Fault-tolerant: can survive crashes, network outages, any single point of failure
HTCondor will ... 

› Keep an eye on your jobs and keep you posted on their progress
› Implement your policy on the execution order of your jobs
› Log your job's activities
› Add fault tolerance to your jobs
› Implement your policy as to when the jobs can run on your desktop
Our esteemed scientist*, has plenty of simulations to do.

* and Karen's cousin
Einstein's Simulation

Simulate the evolution of the cosmos, assuming various properties.
Simulation Overview

Varying values for each of:

- $G$ (the gravitational constant): 100 values
- $R_{\mu\nu}$ (the cosmological constant): 100 values
- $c$ (the speed of light): 100 values

$100 \times 100 \times 100 = 1,000,000$ jobs
Each *job* within the simulation:

- Requires up to 4 GBytes of RAM
- Requires 20 MBytes of input
- Requires 2 – 500 hours of computing time
- Produces up to 10 GBytes of output

Estimated total:

- 15,000,000 CPU hours or 1,700 compute YEARS
- 10 *Petabytes* of output
Albert will be happy, since HTCondor will make the completion of the entire simulation easy.
Definitions

Job
- the HTCondor representation of a piece of work
- Like a Unix process
- Can be an element of a workflow

ClassAd
- HTCondor’s internal data representation

Machine or Resource
- computers that can do the processing
More Definitions

Match Making
- Associating a job with a machine resource

Central Manager
- Central repository for the whole pool
- Does match making

Submit Host
- The computer from which jobs are submitted to HTCondor

Execute Host
- The computer that runs a job
Jobs state their needs and preferences:

- **Requirements** (needs):
  - I require a Linux x86-64 platform

- **Rank** (preferences):
  - I prefer the machine with the most memory
  - I prefer a machine in the botany department
Machines specify needs and preferences:

- **Requirements** (needs):
  - **Require** that jobs run only when there is no keyboard activity
  - **Never** run jobs belonging to Dr. Heisenberg

- **Rank** (preferences):
  - I **prefer** to run Albert’s jobs
ClassAds

the language that Condor uses to represent information – about jobs (job ClassAd), machines (machine ClassAd), and programs that implement Condor's functionality (called daemons)
ClassAd Structure

semi-structured
user-extensible
schema-free

AttributeName = Value
or
AttributeName = Expression
Part of a Job ClassAd

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyType</td>
<td>&quot;Job&quot;</td>
</tr>
<tr>
<td>TargetType</td>
<td>&quot;Machine&quot;</td>
</tr>
<tr>
<td>ClusterId</td>
<td>1</td>
</tr>
<tr>
<td>ProcId</td>
<td>0</td>
</tr>
<tr>
<td>IsPhysics</td>
<td>True</td>
</tr>
<tr>
<td>Owner</td>
<td>&quot;einstein&quot;</td>
</tr>
<tr>
<td>Cmd</td>
<td>&quot;cosmos&quot;</td>
</tr>
<tr>
<td>Requirements</td>
<td>(Arch == &quot;INTEL&quot;)</td>
</tr>
</tbody>
</table>

- **String**
- **Integer**
- **Boolean**
- **Expression**
The Magic of Matchmaking

The **match maker** matches job ClassAds with machine ClassAds, taking into account:

- **Requirements** of both the machine *and* the job
- **Rank** of both the job *and* the machine
- **Priorities**, such as those of users and groups
Getting Started

1. Choose a **universe** for the job
2. Make the job **batch-ready**, which includes making the input data available and accessible
3. Create a **submit description file**
4. Run **condor_submit** to put the job(s) in the queue
1. Choose the **Universe**

- controls how HTCondor handles jobs
- the many universes include:
  - vanilla
  - standard
  - grid
  - java
  - parallel
  - vm
Using the vanilla Universe

• Allows running almost any “serial” job
• Provides automatic file transfer for input and output files
• Like vanilla ice cream, can be used in just about any situation
2. Make the job **batch-ready**

- Must be able to run in the background
- No interactive input
- No GUI/window clicks
Batch-Ready: Standard Input & Output

> Job can still use `stdin`, `stdout` (the keyboard and the screen), and `stderr`, but files are used instead of the actual devices.

> Similar to Unix shell:

```
$ ./myprogram <input.txt >output.txt
```
Make the Data Available

› HTCondor will
  ▶ Transfer data files to the location where the job runs
  ▶ Transfer result files back from the location where the job runs

› Place the job's data files in a place where HTCondor can access them
3. Create a Submit Description File

› A plain ASCII text file
› File name extensions are irrelevant, although many use `.sub` or `.submit` as suffixes
› Tells HTCondor about the job
› Can describe many jobs at once (a cluster), each with different input, output, command line arguments, etc.
Simple Submit Description File

# file name is cosmos.sub
# (Lines beginning with # are comments)
# NOTE: the commands on the left are not
# case sensitive, but file names
# (on the right) are!

Universe   = vanilla
Executable = cosmos
Input      = cosmos.in
Output     = cosmos.out
Log        = cosmos.log

Put 1 instance of the job in the queue
Input, Output, and Error Files

Input = in_file
Read job’s standard input from in_file
Like shell command: $ program < in_file

Output = out_file
Write job’s standard output to out_file
Like shell command: $ program > out_file

Error = error_file
Write job’s standard error to error_file
Like shell command: $ program 2> error_file
Logging the Job's Activities

› In the submit description file:
  \[\text{log} = \text{cosmos.log}\]

› Creates a log of job events, which shows all events that occur as the job executes

› Good advice: *always* have a log file
Sample Portion of Job Log

000 (0101.000.000) 05/25 19:10:03 Job submitted from host: <128.105.146.14:1816>

... 

001 (0101.000.000) 05/25 19:12:17 Job executing on host: <128.105.146.14:1026>

... 

005 (0101.000.000) 05/25 19:13:06 Job terminated.
   (1) Normal termination (return value 0)

... 

000, 001, and 005 are examples of event numbers.
4. Submit the Job

Run `condor_submit`, providing the name of the submit description file:

```
$ condor_submit cosmos.sub
Submitting job(s).
1 job(s) submitted to cluster 100.
```

`condor_submit` then

- parses the submit description file, checking for errors
- creates a ClassAd that describes the job(s)
- places the job(s) in the queue, which is an atomic operation, with a two-phase commit
### Observe Jobs in the Queue

```
$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:51883> : submit.chtc.wisc.edu

<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>2.0</td>
<td>heisenberg</td>
<td>1/13 13:59</td>
<td>0+00:00:00</td>
<td>R</td>
<td>0</td>
<td>0.0</td>
<td>env</td>
</tr>
<tr>
<td>3.0</td>
<td>hawking</td>
<td>1/15 19:18</td>
<td>0+04:29:33</td>
<td>H</td>
<td>0</td>
<td>0.0</td>
<td>script.sh</td>
</tr>
<tr>
<td>4.0</td>
<td>hawking</td>
<td>1/15 19:33</td>
<td>0+00:00:00</td>
<td>H</td>
<td>0</td>
<td>0.0</td>
<td>script.sh</td>
</tr>
<tr>
<td>5.0</td>
<td>hawking</td>
<td>1/15 19:33</td>
<td>0+00:00:00</td>
<td>H</td>
<td>0</td>
<td>0.0</td>
<td>script.sh</td>
</tr>
<tr>
<td>6.0</td>
<td>hawking</td>
<td>1/15 19:34</td>
<td>0+00:00:00</td>
<td>H</td>
<td>0</td>
<td>0.0</td>
<td>script.sh</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96.0</td>
<td>bohr</td>
<td>4/5 13:46</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>atoms H</td>
</tr>
<tr>
<td>97.0</td>
<td>bohr</td>
<td>4/5 13:46</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>atoms H</td>
</tr>
<tr>
<td>98.0</td>
<td>bohr</td>
<td>4/5 13:52</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>atoms H</td>
</tr>
<tr>
<td>99.0</td>
<td>bohr</td>
<td>4/5 13:52</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>atoms H</td>
</tr>
<tr>
<td>100.0</td>
<td>einstein</td>
<td>4/5 13:55</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>cosmos</td>
</tr>
</tbody>
</table>
```

100 jobs; 1 completed, 0 removed, 20 idle, 1 running, 77 held, 0 suspended
File Transfer

Transfer_Input_Files specifies a list of files to transfer from the submit machine to the execute machine.

Transfer_Output_Files specifies a list of files to transfer back from the execute machine to the submit machine. If Transfer_Output_Files is not specified, HTCondor will transfer back all new files in the execute directory.
More on File Transfer

Files need to get from the submit machine to the execute machine. 2 possibilities:

1. both machines have access to a shared file system
2. machines have separate file systems

**Should_Transfer_Files**
- **YES:** Transfer files to execution machine
- **NO:** Rely on shared file system
- **IF_NEEDED:** Automatically transfer the files, if the submit and execute machine are not in the same FileSystemDomain (translation: use shared file system if available)

**When_To_Transfer_Output**
- **ON_EXIT:** Transfer output files only when job completes
- **ON_EXIT_OR_EVICTION:** Transfer output files when job completes or is evicted
File Transfer Example

File Transfer Example

# new cosmos.sub file
Universe = vanilla
Executable = cosmos
Log = cosmos.log
Transfer_Input_Files = cosmos.dat
Transfer_Output_Files = results.dat
Should_Transfer_Files = IF_NEEDED
When_To_Transfer_Output = ON_EXIT
Queue
Command Line Arguments

# Example with command line arguments
Universe   = vanilla
Executable = cosmos
Arguments  = -c 299792458 –G 6.67300e-112

Queue
Invokes executable with
cosmos -c 299792458 -G 6.673e-112

Look at the condor_submit man page to see syntax for Arguments. This example has argc = 5.
More Feedback

- HTCondor sends email about job events to the submitting user
- Specify *one* of these in the submit description file:
  - `Notification = complete`
  - `Notification = never`
  - `Notification = error`
  - `Notification = always`

Default in 7.8

Default in 7.9
ClusterId.ProcID is Job ID

- If the submit description file describes multiple jobs, the set is called a **cluster**
- Each cluster has a **cluster number**, where the cluster number is unique to the job queue on a machine
- Each individual job within a cluster is called a **process**, and **process numbers** always start at zero
- A **Job ID** is the cluster number, a period, and the process number. Examples:

  - Job ID = 20.0  
    - Cluster 20, process 0
  - Job IDs: 21.0, 21.1, 21.2  
    - Cluster 21, processes 0, 1, 2
1 Cluster

Universe = vanilla
Executable = cosmos

log = cosmos_0.log
Input = cosmos_0.in
Output = cosmos_0.out

Queue Job 102.0 (cluster 102, process 0)
File Organization

A logistical nightmare places all input, output, and log files in one directory

3 files × 1,000,000 jobs = 3,000,000 files

- The submit description file is 4,000,000+ lines

The directory will be difficult (at best) to even look at
Better Organization

› Create subdirectories for each job, intentionally named
  run_0, run_1, ... run_999999

› Implement the creation of directories with a Python or Perl program

› Create or place input files in each of these
  run_0/cosmos.in
  run_1/cosmos.in
  ...
  run_999999/cosmos.in

› The output and log files for each job will be created by the job, when the job runs
Einstein’s simulation directory

- cosmos
- cosmos.sub
  - run_θ
    - cosmos.in
    - cosmos.out
    - cosmos.log
  - ...
- run_999999
  - cosmos.in
  - cosmos.out
  - cosmos.log

Submitter or script creates black-font files

HTCondor creates purple-font files
Better Submit Description File

# Cluster of 1,000,000 jobs

Universe = vanilla
Executable = cosmos
Log = cosmos.log
Output = cosmos.out
Input = cosmos.in

...

InitialDir = run_0
Queue

Job 103.0 (Cluster 103, Process 0)

InitialDir = run_1
Queue

Job 103.1 (Cluster 103, Process 1)

This file contains 999,998 more instances of InitialDir and Queue.
Submit Description File Macros

› Queue all 1,000,000 processes with the single command:

   Queue 1000000

› Within the submit description file, HTCondor permits named macros

   $(Process) will be expanded to the process number for each job in the cluster

   Values 0 – 999999 for the 1,000,000 jobs
Using $(Process)$

- The initial directory for each job can be specified as `InitialDir = run_$(Process)`.
  HTCondor expands these to directories: `run_0, run_1, ... run_999999`.

- Similarly, command-line arguments could use a macro to pass a unique identifier to each job instance.
  `Arguments = -n $(Process)`
  HTCondor expands arguments to:
  
  - `-n 0`
  - `-n 1`
  ...
  - `-n 999999`
(Best) Submit Description File

# Example: a cluster of 1000000 jobs

Universe = vanilla
Executable = cosmos
Log = cosmos.log
Input = cosmos.in
Output = cosmos.out
InitialDir = run_${(Process)}
Queue 1000000
Albert submits the cosmos simulation. Patience required, it will take a while...

$ condor_submit cosmos.sub
Submitting
job(s) ..............................................................
..............................................................
..............................................................
..............................................................
..............................................................
Logging submit
event(s) ..............................................................
..............................................................
..............................................................
..............................................................
..............................................................
1000000 job(s) submitted to cluster 104.
the Job Queue

$ condor_q
-- Submitter: submit.chtc.wisc.edu :
   <128.104.55.9:51883> : submit.chtc.wisc.edu

<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>104.0</td>
<td>einstein</td>
<td>4/20 12:08</td>
<td>0+00:00:05</td>
<td>R 0</td>
<td>9.8</td>
<td>cosmos</td>
<td>cosmos</td>
</tr>
<tr>
<td>104.1</td>
<td>einstein</td>
<td>4/20 12:08</td>
<td>0+00:00:03</td>
<td>I 0</td>
<td>9.8</td>
<td>cosmos</td>
<td>cosmos</td>
</tr>
<tr>
<td>104.2</td>
<td>einstein</td>
<td>4/20 12:08</td>
<td>0+00:00:01</td>
<td>I 0</td>
<td>9.8</td>
<td>cosmos</td>
<td>cosmos</td>
</tr>
<tr>
<td>104.3</td>
<td>einstein</td>
<td>4/20 12:08</td>
<td>0+00:00:00</td>
<td>I 0</td>
<td>9.8</td>
<td>cosmos</td>
<td>cosmos</td>
</tr>
</tbody>
</table>

...  
| 104.999998| einstein| 4/20 12:08| 0+00:00:00| I 0| 9.8 | cosmos| cosmos    |
| 104.999999| einstein| 4/20 12:08| 0+00:00:00| I 0| 9.8 | cosmos| cosmos    |

999999 jobs; 999998 idle, 1 running, 0 held
Albert Relaxes

› HTCondor watches over the jobs, and will restart them if required, etc.
› Time for a cold one!
More That You Do With HTCondor
Remove Jobs with `condor_rm`

› You can only remove jobs that you own
› Privileged user can remove any jobs
   - `root` on Linux
   - `administrator` on Windows

`condor_rm 4` Removes all cluster 4 jobs
`condor_rm 4.2` Removes only the job with job ID 4.2
`condor_rm -a` Removes all of your jobs. *Careful!*

---

50
Specify Job Requirements

› A boolean expression (syntax similar to C or Java)
› Evaluated with respect to attributes from machine ClassAd(s)
› **Must** evaluate to True for a match to be made

Universe = vanilla
Executable = mathematica

...

Requirements = ( \n    HasMathematicaInstalled =?= True )
Queue 20
Specify Needed Resources

Items appended to job Requirements

request_memory – the amount of memory (in Mbytes) that the job needs to avoid excessive swapping

request_disk – the amount of disk space (in Kbytes) that the job needs. Will be sum of space for executable, input files, output files and temporary files. Default is size of initial sandbox (executable plus input files).

request_cpus – the number of CPUs (cores) that the job needs. Defaults to 1.
Specify Job Rank

- All matches which meet the requirements can be sorted by preference with a **Rank** expression
  - Numerical
  - Higher rank values match first
- Like **Requirements**, is evaluated against attributes from machine ClassAds

```
Universe       = vanilla
Executable     = cosmos

...```

```
Rank            = (KFLOPS*10000) + Memory
Queue 1000000
```
Job Policy Expressions

- Do not remove if exits with a signal:
  
  ```python
  on_exit_remove = ExitBySignal == False
  ```

- Place on hold if exits with nonzero status or ran for less than an hour:
  
  ```python
  on_exit_hold =
  ( (ExitBySignal==False) && (ExitSignal != 0) ) ||
  ( (ServerStartTime - JobStartDate) < 3600)
  ```

- Place on hold if job has spent more than 50% of its time suspended:
  
  ```python
  periodic_hold =
  ( CumulativeSuspensionTime >
    (RemoteWallClockTime / 2.0) )
  ```
Lots of Short-Running Jobs

Know that starting a job is somewhat expensive, in terms of time.

3 items that might help:

1. Batch short jobs together
   - Write a wrapper script that will run a set of the jobs in series
   - Submit the wrapper script as the job

2. Explore HTCondor’s parallel universe

3. There are some configuration variables that may be able to help
   - Contact a staff person for more info
Common Problems with Jobs
Jobs Are Idle

Our scientist runs `condor_q` and finds all his jobs are idle

```bash
$ condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510>
 :x.cs.wisc.edu

<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>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.1</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.2</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.3</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.4</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.5</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.6</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
<tr>
<td>5.7</td>
<td>einstein</td>
<td>4/20 12:23</td>
<td>0+00:00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>cosmos</td>
</tr>
</tbody>
</table>

8 jobs; 8 idle, 0 running, 0 held
Exercise a little patience

➢ On a busy pool, it can take a while to match jobs to machines, and then start the jobs

➢ Wait at least a negotiation cycle or two, typically a few minutes
Look in the Job Log

The log will likely contain clues:

```
$ cat cosmos.log
000 (031.000.000) 04/20 14:47:31 Job submitted from host: <128.105.121.53:48740>
...
007 (031.000.000) 04/20 15:02:00 Shadow exception!
    Error from starter on gig06.stat.wisc.edu:
    Failed to open '/scratch.1/einstein/workspace/v78/condor-test/test3/run_0/cosmos.in' as standard input: No such file or directory (errno 2)
    0  -  Run Bytes Sent By Job
    0  -  Run Bytes Received By Job
...
```
Check Machines' Status

$ condor_status

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActvtyTime</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:slot1@c002.chtc.wi">slot1@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>4599</td>
<td>0+00:10:13</td>
</tr>
<tr>
<td><a href="mailto:slot2@c002.chtc.wi">slot2@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>1024</td>
<td>1+19:10:36</td>
</tr>
<tr>
<td><a href="mailto:slot3@c002.chtc.wi">slot3@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>0.990</td>
<td>1024</td>
<td>1+22:42:20</td>
</tr>
<tr>
<td><a href="mailto:slot4@c002.chtc.wi">slot4@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>1024</td>
<td>0+03:22:10</td>
</tr>
<tr>
<td><a href="mailto:slot5@c002.chtc.wi">slot5@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>1024</td>
<td>0+03:17:00</td>
</tr>
<tr>
<td><a href="mailto:slot6@c002.chtc.wi">slot6@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>1024</td>
<td>0+03:09:14</td>
</tr>
<tr>
<td><a href="mailto:slot7@c002.chtc.wi">slot7@c002.chtc.wi</a></td>
<td>LINUX</td>
<td>X86_64</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.000</td>
<td>1024</td>
<td>0+19:13:49</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slot7@exec-2.chtc">slot7@exec-2.chtc</a>.</td>
<td>WINDOWS</td>
<td>INTEL</td>
<td>Owner</td>
<td>Idle</td>
<td>0.000</td>
<td>511</td>
<td>0+00:24:17</td>
</tr>
<tr>
<td><a href="mailto:slot8@exec-2.chtc">slot8@exec-2.chtc</a>.</td>
<td>WINDOWS</td>
<td>INTEL</td>
<td>Owner</td>
<td>Idle</td>
<td>0.030</td>
<td>511</td>
<td>0+00:45:01</td>
</tr>
</tbody>
</table>

Total Owner Claimed Unclaimed Matched Preempting Backfill

<table>
<thead>
<tr>
<th></th>
<th>Owner</th>
<th>Claimed</th>
<th>Unclaimed</th>
<th>Matched</th>
<th>Preempting</th>
<th>Backfill</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL/WINDOWS</td>
<td>104</td>
<td>78</td>
<td>16</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X86_64/LINUX</td>
<td>759</td>
<td>170</td>
<td>587</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>863</td>
<td>248</td>
<td>603</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Try: condor_q -analyze

> condor_q -analyze 107.5
-- Submitter: crane.cs.wisc.edu :
   <128.105.136.32:61610> : crane.cs.wisc.edu
User priority for max@crane.cs.wisc.edu is not available, attempting to analyze without it.
---
107.005: Run analysis summary. Of 4 machines,
  0 are rejected by your job's requirements
  0 reject your job because of their own requirements
  4 match and are already running your jobs
  0 match but are serving other users
  0 are available to run your job

improved in 7.9
condor_q -analyze 102.1

-- Submitter: crane.cs.wisc.edu:
  <128.105.136.32:61610> : crane.cs.wisc.edu
  User priority for max@crane.cs.wisc.edu is not available, attempting to analyze without it.

---

107.005: Run analysis summary. Of 3184 machines, 3184 are rejected by your job's requirements
  0 reject your job because of their own requirements
  0 match and are already running your jobs
  0 match but are serving other users
  0 are available to run your job

WARNING: Be advised:
  No resources matched request's constraints
The Requirements expression for your job is:

```
( TARGET.Arch == "X86_64" ) &&
( TARGET.OpSys == "WINDOWS" ) &&
( TARGET.Disk >= RequestDisk ) &&
( TARGET.Memory >= RequestMemory ) &&
( TARGET.HasFileTransfer )
```

Suggestions:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Machines Matched</th>
<th>Suggestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( TARGET.OpSys == &quot;WINDOWS&quot; )</td>
<td>0</td>
<td>MODIFY TO &quot;LINUX&quot;</td>
</tr>
<tr>
<td>2 ( TARGET.Arch == &quot;X86_64&quot; )</td>
<td>3137</td>
<td></td>
</tr>
<tr>
<td>3 ( TARGET.Disk &gt;= 1 )</td>
<td>3184</td>
<td></td>
</tr>
<tr>
<td>4 ( TARGET.Memory &gt;= ifthenelse(MemoryUsage isnt undefined, MemoryUsage, 1) )</td>
<td>3184</td>
<td></td>
</tr>
<tr>
<td>5 ( TARGET.HasFileTransfer )</td>
<td>3184</td>
<td></td>
</tr>
</tbody>
</table>
Learn about available resources

```
$ condor_status -const 'Memory > 8192'
(no output means no matches)

$ condor_status -const 'Memory > 4096'
Name          OpSys  Arch   State     Activ LoadAv Mem   ActvtyTime
slot1@c001.ch LINUX X86_64 Unclaimed Idle  0.000  5980  1+05:35:05
slot2@c001.ch LINUX X86_64 Unclaimed Idle  0.000  5980 13+05:37:03
slot3@c001.ch LINUX X86_64 Unclaimed Idle  0.000  7988  1+06:00:05
slot1@c002.ch LINUX X86_64 Unclaimed Idle  0.000  7988 13+06:03:47

Total Owner Claimed Unclaimed Matched Preempting
X86_64/LINUX     4     0       0         4       0          0
Total           4     0       0         4       0          0
```
Interact With A Job

› Perhaps a job is running for much longer than expected.
   Is it stuck accessing a file?
   ▷ Is it in an infinite loop?

› Try `condor_ssh_to_job`
   ▷ Interactive debugging in Unix
   ▷ Use `ps`, `top`, `gdb`, `strace`, `lsof`, …
   ▷ Forward ports, X, transfer files, etc.
   ▷ Currently not available on Windows
Interactive Debug Example

$ condor_q
-- Submitter: cosmos.phy.wisc.edu : <128.105.165.34:1027>

ID OWNER SUBMITTED RUN_TIME ST PRI SIZE CMD
1.0 einstein 4/15 06:52 1+12:10:05 R 0 10.0 cosmos

1 jobs; 0 idle, 1 running, 0 held

$ condor_ssh_to_job 1.0

Welcome to slot4@c025.chtc.wisc.edu!
Your condor job is running with pid(s) 15603.

$ gdb -p 15603

...
HTCondor is extremely flexible. Here are overviews of some of the many features that you may want to learn more about.
After this tutorial, here are some places you might find help:

1. HTCondor manual
2. htcondor-users mailing list. See https://lists.cs.wisc.edu/mailman/listinfo/htcondor-users
3. Wiki
   https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki
4. Developers
The more time a job takes to run, the higher the risk of

- being **preempted** by a higher priority user or job
- getting kicked off a machine (**vacated**), because the machine has something else it prefers to do

- HTCondor's **standard universe** may provide a solution.
Standard Universe

› Regularly while the job runs, or when the job is to be kicked off the machine, HTCondor takes a **checkpoint** -- the complete state of the job.

› With a checkpoint, the job can be matched to another machine, and *continue on*. 
checkpoint: the entire state of a program saved in a file, such as CPU registers, memory image, I/O, etc.
3 Checkpoints
Goodput and Badput

![Diagram showing goodput and badput over time with a value of 3]
Standard Universe Features

› Remote system calls (remote I/O)
  The job can read or write files as if they were local
› Programming language independent
› No source code changes are typically required, but relinking the executable with HTCondor's standard universe support library is required.
How to Relink

Place `condor_compile` in front of the command used to link the job:

```bash
$ condor_compile gcc -o myjob myjob.c
- OR -
$ condor_compile f77 -o myjob filea.f fileb.f
- OR -
$ condor_compile make -f MyMakefile
```
Limitations

› HTCondor’s checkpoint mechanism is not at the kernel level. Therefore, a standard universe job may not:
  fork()
  ● Use kernel threads
  ● Use some forms of IPC, such as pipes and shared memory
› Must have access to object code in order to relink
› Only available on some Linux platforms
Parallel Universe

› When multiple processes of a single job must be running at the same time on different machines.

› Provides a mechanism for controlling parallel algorithms
  - Fault tolerant
    - Allows for resources to come and go
    - Ideal for Computational Grid environments

› Especially for MPI
# MPI job submit description file
universe = parallel
executable = mp1script
arguments = my_mpich_linked_exe arg1 arg2
machine_count = 4
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = my_mpich_linked_exe
queue
MPI jobs

Note: HTCondor will probably not schedule all of the jobs on the same machine, so consider using **whole machine slots**

See the HTCondor Wiki:
Under *How To Admin Recipes*,
"How to allow some jobs to claim the whole machine instead of one slot"
VM Universe

- A virtual machine instance is the HTCondor job
- The vm universe offers
  - Job sandboxing
  - Checkpoint and migration
  - Safe elevation of privileges
  - Cross-platform submission
- HTCondor supports VMware, Xen, and KVM
- Input files can be imported as CD-ROM image
- When the VM shuts down, the modified disk image is returned as job output
Machine Resources are Numerous: The Grid

Given access (authorization) to grid resources, as well as certificates (for authentication) and access to Globus or other resources at remote institutions, HTCondor's grid universe does the trick!
Grid Universe

- All specification is in the submit description file
- Supports many “back end” types:
  - Globus: GT2, GT5
  - NorduGrid
  - UNICORE
  - HTCondor
  - PBS
  - LSF
  - SGE
  - EC2
  - Deltacloud
  - Cream
Some sets of jobs have dependencies. HTCondor handles them with DAGMan.

› Interested? Stay for Kent's tutorial, later this morning.
the Java Universe

More than

$ java mysimulator

› Knows which machines have a JVM installed
› Knows the location, version, and performance of JVM on each machine
› Knows about jar files, etc.
› Provides more information about Java job completion than just a JVM exit code
  ▷ Program runs in a Java wrapper, allowing HTCondor to report Java exceptions, etc.
Java Universe Example

# sample java universe submit
# description file
Universe = java
Executable = Main.class
jar_files = MyLibrary.jar
Input = infile
Output = outfile
Arguments = Main 1 2 3
Queue
In Review

With HTCondor’s help, both you and Albert can:

- Submit jobs
- Manage jobs
- Organize data files
- Identify aspects of universe choice
Thank you!

Check us out on the web:
http://www.research.wisc.edu/htcondor

Email:
htcondor-admin@cs.wisc.edu
Extra Slides with More Information You Might Want to Reference
InitialDir

› Identifies a directory for file input and output.
› Also provides a directory (on the submit machine) for the user log, when a full path is not specified.
› **Note:** Executable is **not** relative to InitialDir

```bash
# Example with InitialDir
Universe   = vanilla
InitialDir = /home/einstein/cosmos/run
Executable = cosmos
Log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
Error      = cosmos.err
Transfer_Input_Files=cosmos.dat
Arguments  = -f cosmos.dat
Queue
```

`NOT Relative to InitialDir`

`Is Relative to InitialDir`
Substitution Macro

$$\langle\text{attribute}\rangle$$ will be replaced by the value of the specified attribute from the Machine ClassAd

Example:

Machine ClassAd has:

\textbf{CosmosData} = "/local/cosmos/data"

Submit description file has

\textbf{Executable} \quad = \quad \text{cosmos}
\textbf{Requirements} \quad = \quad (\text{CosmosData} \neq \text{UNDEFINED})
\textbf{Arguments} \quad = \quad -d \quad $$\langle\text{CosmosData}\rangle$

Results in the job invocation:

\text{cosmos} \ -d \quad /local/cosmos/data
Getting HTCondor

› Available as a free download from http://research.cs.wisc.edu/htcondor

› Download HTCondor for your operating system
   - Available for many modern Unix platforms (including Linux and Apple’s OS/X)
   - Also for Windows, many versions

› Repositories
   - YUM: RHEL 4, 5, and 6
     - $ yum install condor
   - APT: Debian 5 and 6
     - $ apt-get install condor
HTCondor Releases

› Stable and Developer Releases
  ▷ Version numbering scheme similar to that of the (pre 2.6) Linux kernels ...

› Major.minor.release
  ▷ If minor is even (a.b.c): Stable series
    • Very stable, mostly bug fixes
    • Current: 7.8
  ▷ If minor is odd (a.b.c): Developer series
    • New features, may have some bugs
    • Current: 7.9
General User Commands

- condor_status: View Pool Status
- condor_q: View Job Queue
- condor_submit: Submit new Jobs
- condor_rm: Remove Jobs
- condor_prio: Change a User Priority
- condor_history: Completed Job Info
- condor_submit_dag: Submit new DAG
- condor_checkpoint: Force a checkpoint
- condor_compile: Link Condor library with job