

More HTCondor for HTC

Monday AM, Lecture 2

Lauren Michael

Questions so far?

Goals for this Session

- Understand HTCondor mechanisms more deeply
- Best ways to submit multiple jobs (what we're here for, right?)
- Testing and troubleshooting
- Automation, additional use cases, and features

How is HTC Optimized?

- System must track jobs, machines, policy, ...
- System must recover gracefully from failures
- Try to use all available resources, all the time
- Lots of variety in users, machines, networks,
- ...
- Sharing is hard (e.g. policy, security)

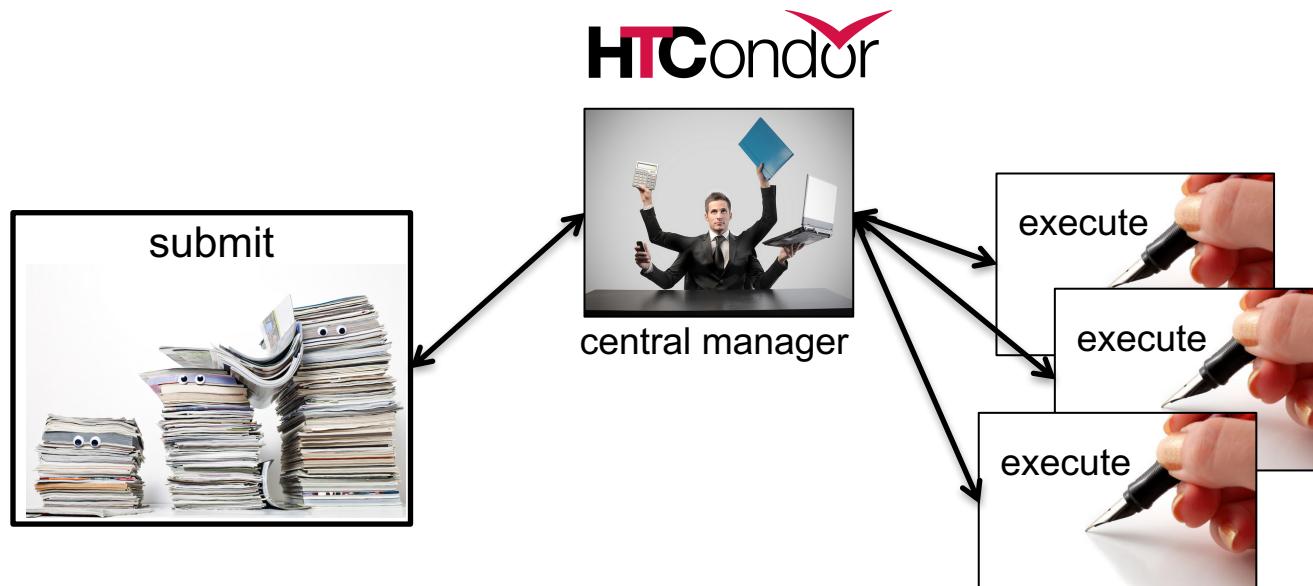
HTCONDOR MATCHMAKING

Roles in an HTCondor System

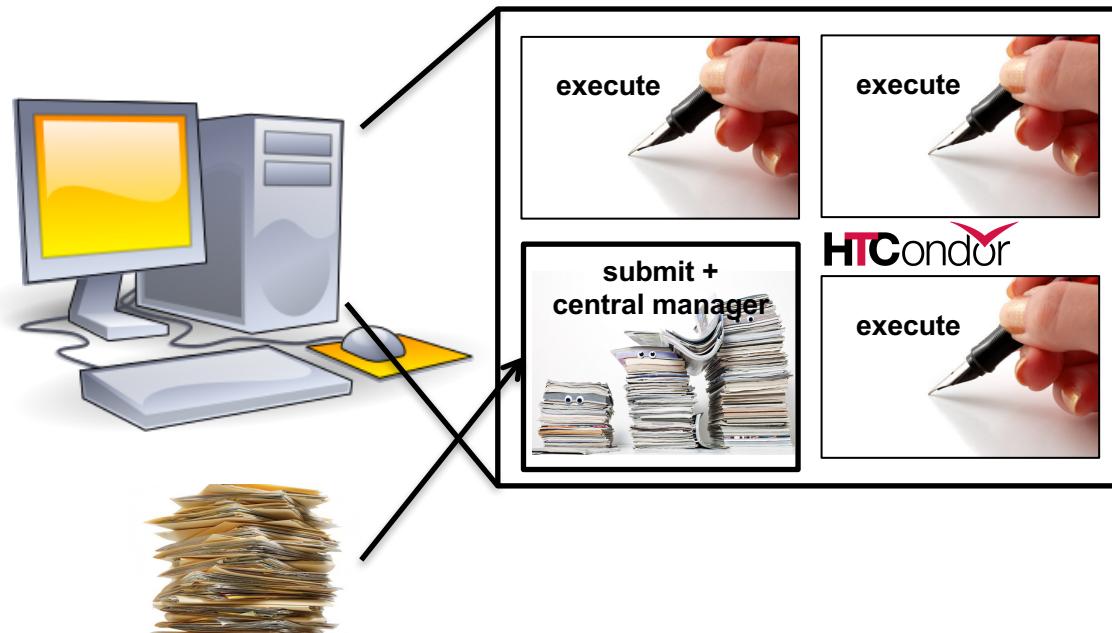
- **Users**
 - Define jobs, their requirements, and preferences
 - Submit and cancel jobs
 - Check on the status of jobs
- **Administrators**
 - Configure and control the HTCondor system
 - Implement policies
 - Check on the status of machines
- **HTCondor Software**
 - Track and manage machines
 - Track and run jobs
 - Match jobs to machines (enforcing all policies)

Job Matching

- On a regular basis, the **central manager** reviews **Job** and **Machine** attributes, and pool policies, and matches jobs to **slots**.



Single Computer



Terminology: Matchmaking

two-way process of finding a slot for a job

- ***Jobs have requirements and preferences***
 - e.g.: I need one CPU core, 100 GB of disk space, and 10 GB of memory
- ***Machines have requirements and preferences***
 - E.g.: I run jobs only from users in the Comp. Sci. dept., and prefer to run ones that ask for a lot of memory
- ***Important jobs may run first or replace less important ones***

HTCondor Priorities

- **User priority**
 - Computed based on past usage
 - Determines user's "fair share" percentage of slots
 - Lower number means run sooner (0.5 is minimum)
- **Job priority**
 - Set per job by the user (owner)
 - Relative to that user's other jobs
 - Set in submit file or changed later with `condor_prio`
 - Higher number means run sooner
- **Preemption**
 - Low priority jobs stopped for high priority ones (stopped jobs go back into the regular queue)
 - Governed by fair-share algorithm and pool policy
 - Not enabled on all pools

Class Ads

- HTCondor stores a list of information about **each job and each machine** of potential slots.
- This information is stored for each job and each machine as its “**Class Ad**”



- Class Ads have the format:
`AttributeName = value`

can be a boolean (T/F),
number, or string

Job ClassAd

Submit file

```
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

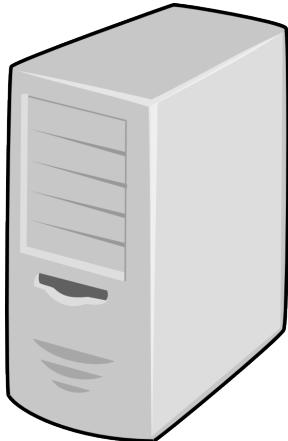
queue 1
```

+

Default HTCondor
configuration

```
=
RequestCpus = 1
Err = "job.err"
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd =
"/home/alice/tests/htcondor_week/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week"
NumJobStarts = 0
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job"
Out = "job.out"
UserLog =
"/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...
```

Machine ClassAd



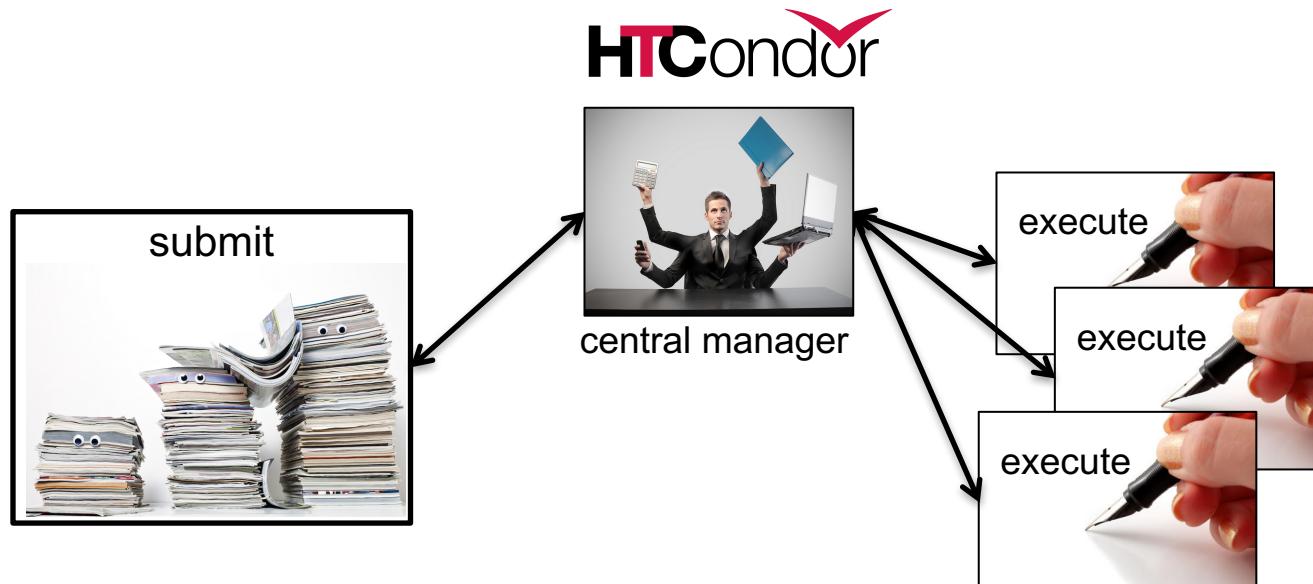
=

+
Default 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
...
```

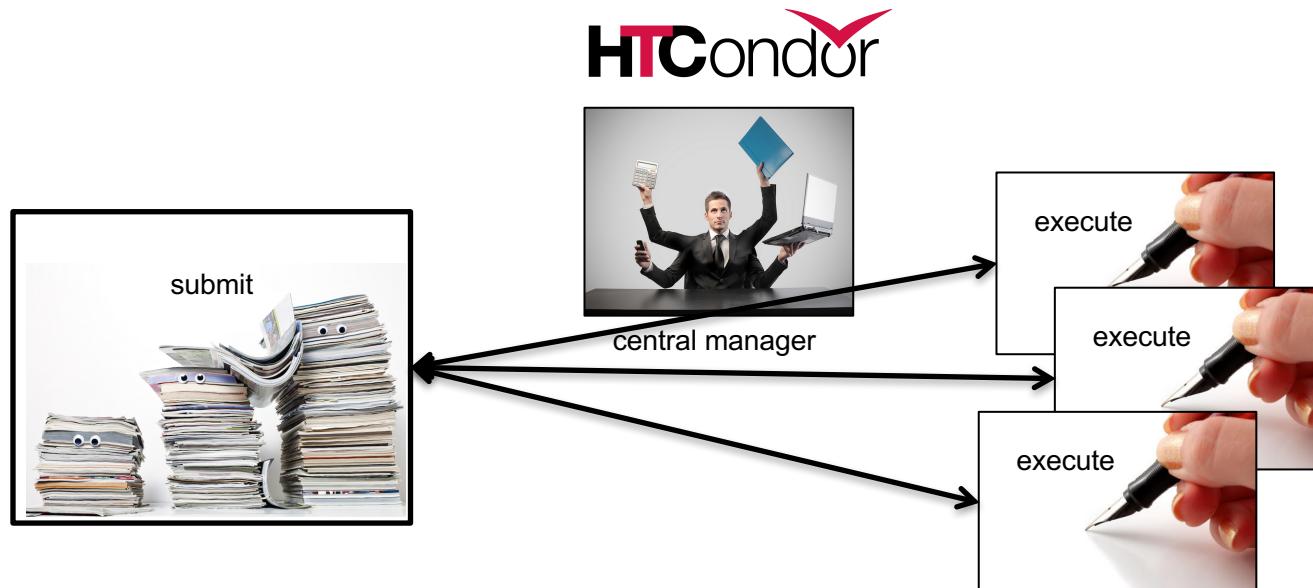
Job Matching

- On a regular basis, the central manager reviews **Job** and **Machine ClassAds** and matches jobs to **slots**.



Job Execution

- (Then the submit and execute points communicate directly.)



USING CLASSADS

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 12008.0
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
- **JobBatchName**: user-labeled job batches
- ...and more



Displaying Job Attributes

- View only specific attributes (**-af** for ‘autoformat’)

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

```
$ condor_q -af ClusterId ProcId RemoteHost MemoryUsage

17315225 116 slot1_1@e092.ctc.wisc.edu 1709
17315225 118 slot1_2@e093.ctc.wisc.edu 1709
17315225 137 slot1_8@e125.ctc.wisc.edu 1709
17315225 139 slot1_7@e121.ctc.wisc.edu 1709
18050961 0 slot1_5@c025.ctc.wisc.edu 196
18050963 0 slot1_3@atlas10.ctc.wisc.edu 269
18050964 0 slot1_25@e348.ctc.wisc.edu 245
```

condor_q Reminder

- Default output is batched jobs
 - Batches can be grouped by the user with the **JobBatchName** attribute in a submit file:

```
JobBatchName = CoolJobs
```
 - Otherwise HTCondor groups jobs, automatically, by same executable
- To see individual jobs, use:

condor_q -nobatch

ClassAds for Machines & Slots

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

```
$ 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

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	0

Total	10964	2	10340	613	0	0	0	9
-------	-------	---	-------	-----	---	---	---	---

Machine Attributes

- Use same ClassAd options as **condor_q**:

condor_status -l Slot/Machine

condor_status [Machine] -af Attribute1 Attribute2 ...

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

Machine Attributes

- To summarize, use the “-compact” option:
- condor_status -compact**

```
$ condor_q -compact
Machine          Platform   Slots  Cpus  Gpus  TotalGb  FreCpu  FreeGb  CpuLoad ST
e007.chtc.wisc.edu  x64/SL6     8      8       23.46    0        0.00   1.24  Cb
e008.chtc.wisc.edu  x64/SL6     8      8       23.46    0        0.46   0.97  Cb
e009.chtc.wisc.edu  x64/SL6    11     16       23.46    5        0.00   0.81  **
e010.chtc.wisc.edu  x64/SL6     8      8       23.46    0        4.46   0.76  Cb
matlab-build-1.chtc.wisc.edu  x64/SL6    1     12       23.45   11       13.45  0.00  **
matlab-build-5.chtc.wisc.edu  x64/SL6     0     24       23.45   24       23.45  0.04  Ui
mem1.chtc.wisc.edu    x64/SL6    24     80      1009.67   8        0.17   0.60  **

Total  Owner  Claimed  Unclaimed  Matched  Preempting  Backfill  Drain
x64/SL6  10416    0    9984      427      0          0        0        5
x64/WinVista  2      2      0          0      0          0        0        0

Total 10418    2    9984      427      0          0        0        5
```

SUBMITTING MULTIPLE JOBS

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
 - scale up by breaking up!
 - *we're learning HTC, right?*
- ...without having to:
 - create separate submit files for each job
 - submit and monitor each job, individually

From one job ...

```
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
```

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

Multiple numbered input files

```
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 3
```

```
(submit_dir)/
```

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

```
job.submit
```

- Generates 3 jobs, but doesn't change inputs and will overwrite the outputs
- So how can we specify different values to each job?

Manual Approach (Not recommended!)

```
job.submit
```

```
executable = analyze.exe
log = job.log

arguments = file0.in file0.out
transfer_input_files = file0.in
output = job0.out
error = job0.err
queue 1

arguments = file1.in file1.out
transfer_input_files = file1.in
output = job1.out
error = job1.err
queue 1
```

```
...
```

```
(submit_dir)/
```

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

```
job.submit
```

Automatic Variables

	ClusterId	ProcId	
queue N	128	0	Each job's ClusterId and ProcId numbers are autogenerated and saved as job attributes
	128	1	
	128	2	
	
	128	$N-1$	They can be referenced inside the submit file using: – <code>\$(ClusterId)</code> – <code>\$(ProcId)</code>

Using \$(Procl) for Numbered Files

job.submit

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

log = job_$(Cluster).log
output = job_$(Process).out
error = job_$(Process).err
```

```
queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in
job.submit
```

- \$(Process) and \$(Cluster) allow us to provide unique values to each job and submission!

Organizing Files in Sub-Directories

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



* must be created before the job is submitted

Shared Files

- HTCondor can transfer an entire directory or all the contents of a directory
 - transfer whole directory

```
transfer_input_files = shared
```
 - transfer contents only

```
transfer_input_files = shared/
```
- Useful for jobs with many shared files; transfer a directory of files instead of listing files individually

```
(submit_dir)/  
job.submit  
shared/  
reference.db  
parse.py  
analyze.py  
cleanup.py  
links.config
```

Use Paths for File Type

(submit_dir)/

		input/	log/	err/
job.submit	file0.out	file0.in	job0.log	job0.err
analyze.exe	file1.out	file1.in	job1.log	job1.err
	file2.out	file2.in	job2.log	job2.err

job.submit

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

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

queue 3
```

Separating Files by Job with InitialDir

- **Initialdir** sets the initial location for each job's files, allowing each job to "live" in separate directories on the submit server
- Allows same filenames for input/output files across jobs
- Also useful for jobs with lots of output files



Separating jobs with initialdir

(submit_dir)/

job.submit
analyze.exe

job0/
file.in
job.log
job.err
file.out

job1/
file.in
job.log
job.err
file.out

job2/
file.in
job.log
job.err
file.out

job.submit

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

log = job.log
error = job.err

queue 3
```

executable must be relative
to the submission directory,
and *not* in the InitialDir.

Many jobs per submit file

- Back to our compare_states example...
- What if we had data for each state? We could do 50 submit files (or 50 “queue 1” statements) ...

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

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

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

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

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

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

Many jobs per submit file

- Back to our compare_states example...
- What if we had data for each state? We could do 50 submit files (or 50 “queue 1” statements) ...

```
executable = compare_states
arguments = vt.dat us.dat vt.dat.out
arguments = wa.dat us.dat wa.dat.out
arguments = vi.dat us.dat vi.dat.out
...
executable = compare_states
arguments = tx.dat us.dat tx.dat.out
arguments = mi.dat us.dat mi.dat.out
arguments = mu.dat us.dat mu.dat.out
...
executable = compare_states
arguments = ak.dat us.dat ak.dat.out
arguments = sd.dat us.dat sd.dat.out
arguments = wv.dat us.dat wv.dat.out
...
...
```

```
executable = compare_states
arguments = al.dat us.dat al.dat.out
arguments = co.dat us.dat co.dat.out
arguments = ba.dat us.dat ba.dat.out
...
...
executable = compare_states
arguments = ut.dat us.dat ut.dat.out
arguments = nv.dat us.dat nv.dat.out
arguments = nv.dat us.dat nv.dat.out
...
...
executable = compare_states
arguments = tn.dat us.dat tn.dat.out
arguments = mn.dat us.dat mn.dat.out
arguments = nn.dat us.dat nn.dat.out
...
...

```

Many jobs per submit file

- We could rename (map) our data to fit the \$(Process) or approach ...
- Or we could use HTCondor's powerful **queue** language to submit jobs using our own variables!

Submitting Multiple Jobs – Queue Statements

multiple “queue” statements	<pre>state = wi.dat queue 1 state = ca.dat queue 1 state = mo.dat queue 1</pre>	Not Recommended
var matching <i>pattern</i>	<pre>queue state matching *.dat</pre>	
var in (i ii iii ...)	<pre>queue state in (wi.dat ca.dat co.dat)</pre>	
var1,var2 from csv_file	<pre>queue state from state_list.txt</pre>	<pre>state_list.txt: wi.dat ca.dat mo.dat ...</pre>

Using Multiple Variables

- Both the “from” and “in” syntax support multiple variables from a list.

job.submit

```
executable = compare_states
arguments = -y $(year) -i $(infile)

transfer_input_files = $(infile)

queue infile,year from job_list.txt
```

job_list.csv

```
wi.dat, 2010
wi.dat, 2015
ca.dat, 2010
ca.dat, 2015
mo.dat, 2010
mo.dat, 2015
```

Submitting Multiple Jobs – Queue Statements

multiple “queue” statements	Not recommended. Can be useful when submitting job batches where a single non-file/non-argument characteristic is changing
var matching pattern	Natural nested looping, minimal programming, can use “files” or “dirs” keywords to narrow possible matches. Requires good naming conventions, less reproducible.
var in (i,ii,iii,...)	All information contained in the submit file: reproducible. Harder to automate submit file creation.
var1,var2 from csv_file	Supports multiple variables, highly modular (easy to use one submit file for many job batches that have different var lists), reproducible. Additional file needed, but can be automated.

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
```

- Combine with InitialDir:

```
InitialDir = $(directory)
```

```
queue directory matching dirs job*
```

TESTING AND TROUBLESHOOTING

What Can Go Wrong?

- Jobs can go wrong “internally”:
 - the executable experiences an error
- Jobs can go wrong from HTCondor’s perspective:
 - a job can’t be matched
 - a job is missing files
 - 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

Log	Output	Error
<ul style="list-style-type: none">• When jobs were submitted, started, held, or stopped• Resources used• Exit status• Where job ran• Interruption reasons	Any “print” or “display” information from your program (may contain errors from the executable).	Errors captured by the operating system while the executable ran, or reported by the executable, itself.

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
  ID      OWNER      SUBMITTED      RUN_TIME      ST      COMPLETED      CMD
189.1012  alice  5/11 09:52  0+00:07:37  C  5/11 16:00 /home/alice
189.1002  alice  5/11 09:52  0+00:08:03  C  5/11 16:00 /home/alice
189.1081  alice  5/11 09:52  0+00:03:16  C  5/11 16:00 /home/alice
189.944   alice  5/11 09:52  0+00:11:15  C  5/11 16:00 /home/alice
189.659   alice  5/11 09:52  0+00:26:56  C  5/11 16:00 /home/alice
189.653   alice  5/11 09:52  0+00:27:07  C  5/11 16:00 /home/alice
189.1040  alice  5/11 09:52  0+00:05:15  C  5/11 15:59 /home/alice
189.1003  alice  5/11 09:52  0+00:07:38  C  5/11 15:59 /home/alice
189.962   alice  5/11 09:52  0+00:09:36  C  5/11 15:59 /home/alice
189.961   alice  5/11 09:52  0+00:09:43  C  5/11 15:59 /home/alice
189.898   alice  5/11 09:52  0+00:13:47  C  5/11 15:59 /home/alice
```

“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.
 - files not found for transfer, over memory, etc.
- 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 until removed, or (fixed and) released.



Diagnosing Holds

- If HTCondor puts a job on hold, it provides a hold reason, which can be viewed in the log file, or with:

```
condor_q -hold -af HoldReason
```

```
$ condor_q -hold -af HoldReason
Error from slot1_1@wid-003.chtc.wisc.edu: Job has gone over
memory limit of 2048 megabytes.
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>
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
```



Common Hold Reasons

- Job has used **more memory** than requested.
- **Incorrect path to files** that need to be transferred
- **Badly formatted executable scripts** (have Windows instead of Unix line endings)
- Submit directory is **over quota**.
- **Job has run for too long.** (72 hours allowed in CHTC Pool)
- The **admin has put your job on hold**.

Fixing Holds

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

condor_qedit [U/C/J] Attribute Value

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

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

condor_release [U/C/J]

```
$ condor_release 128.0  
Job 18933774.0 released
```

Holding or Removing Jobs

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

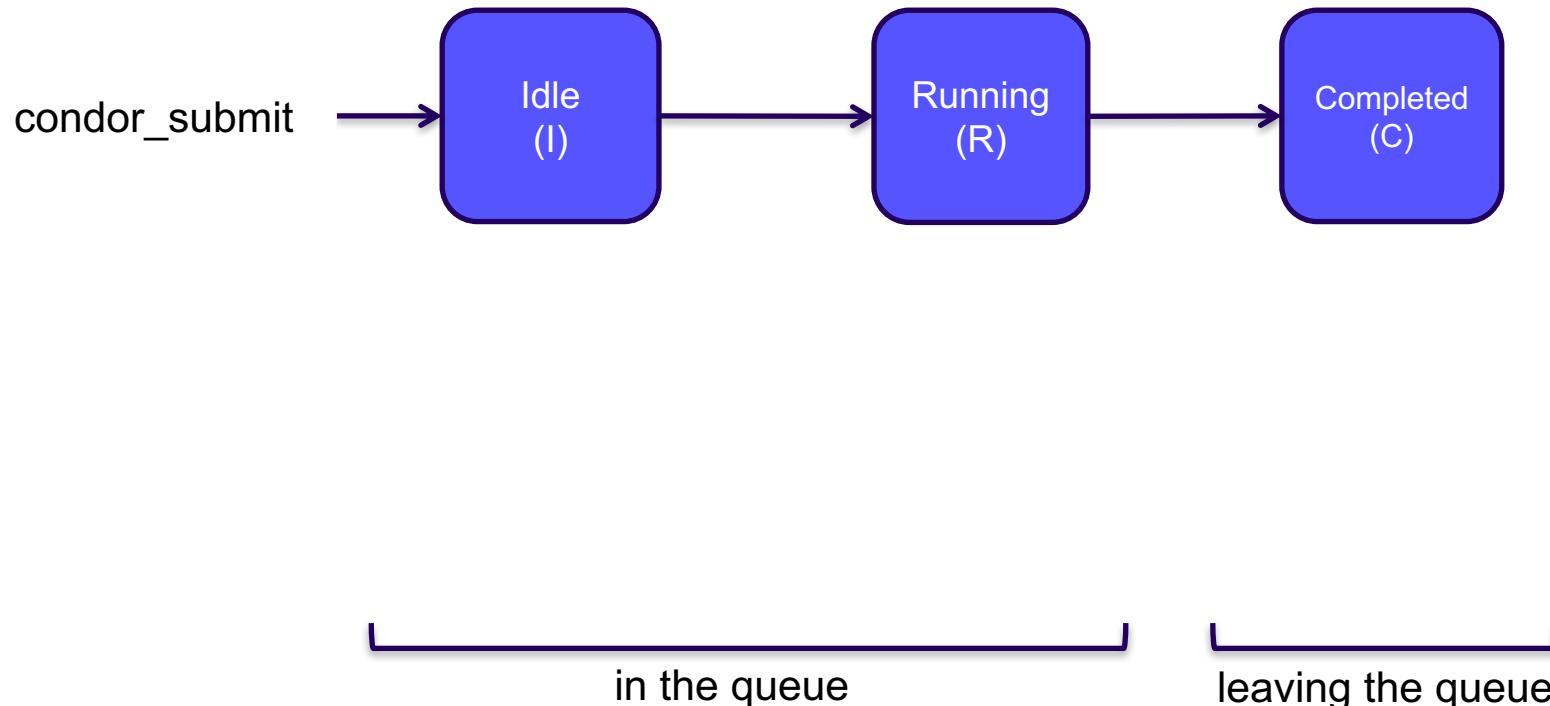
```
$ condor_hold bob  
All jobs of user "bob" have been held
```

```
$ condor_hold 128  
All jobs in cluster 128 have been held
```

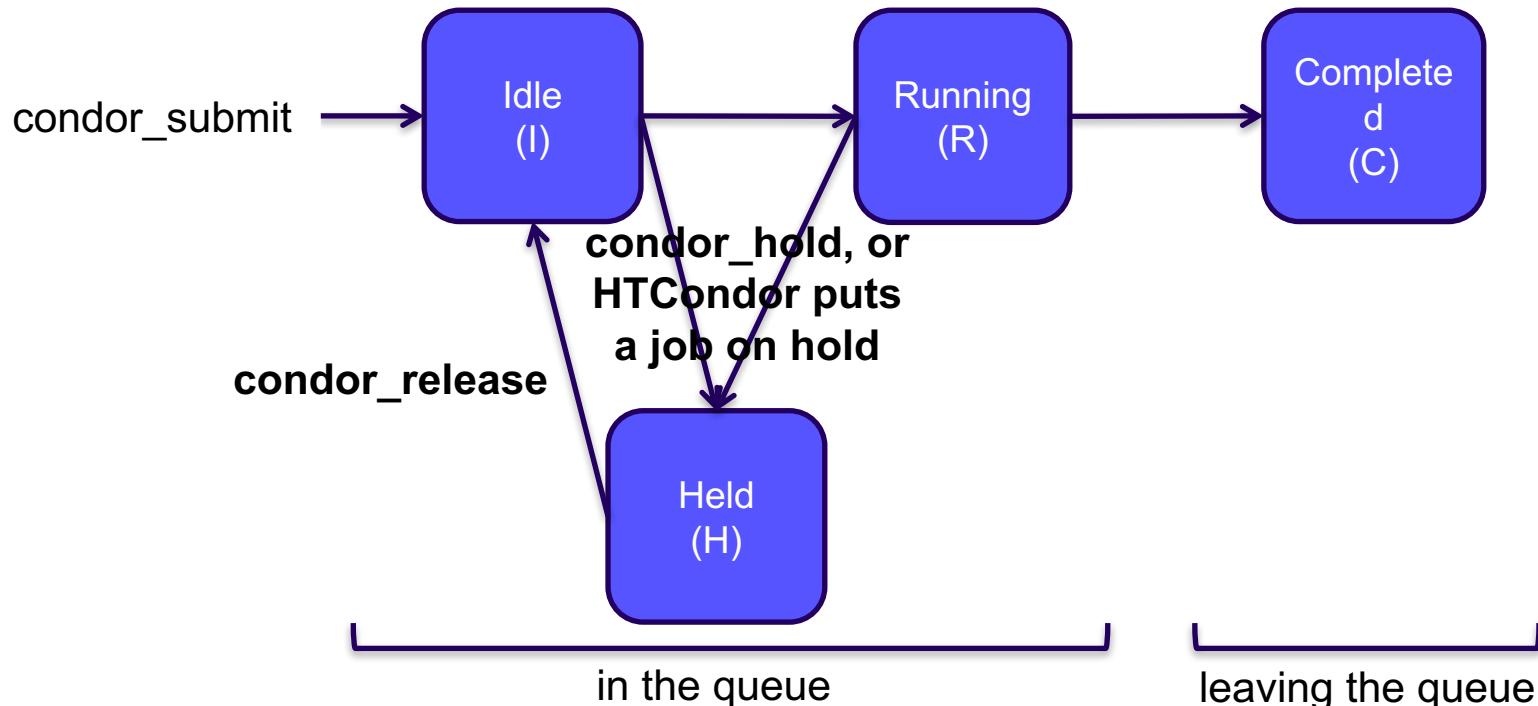
```
$ condor_hold 128.0  
Job 128.0 held
```

- Remove it from the queue, using **condor_rm [U/C/J]**

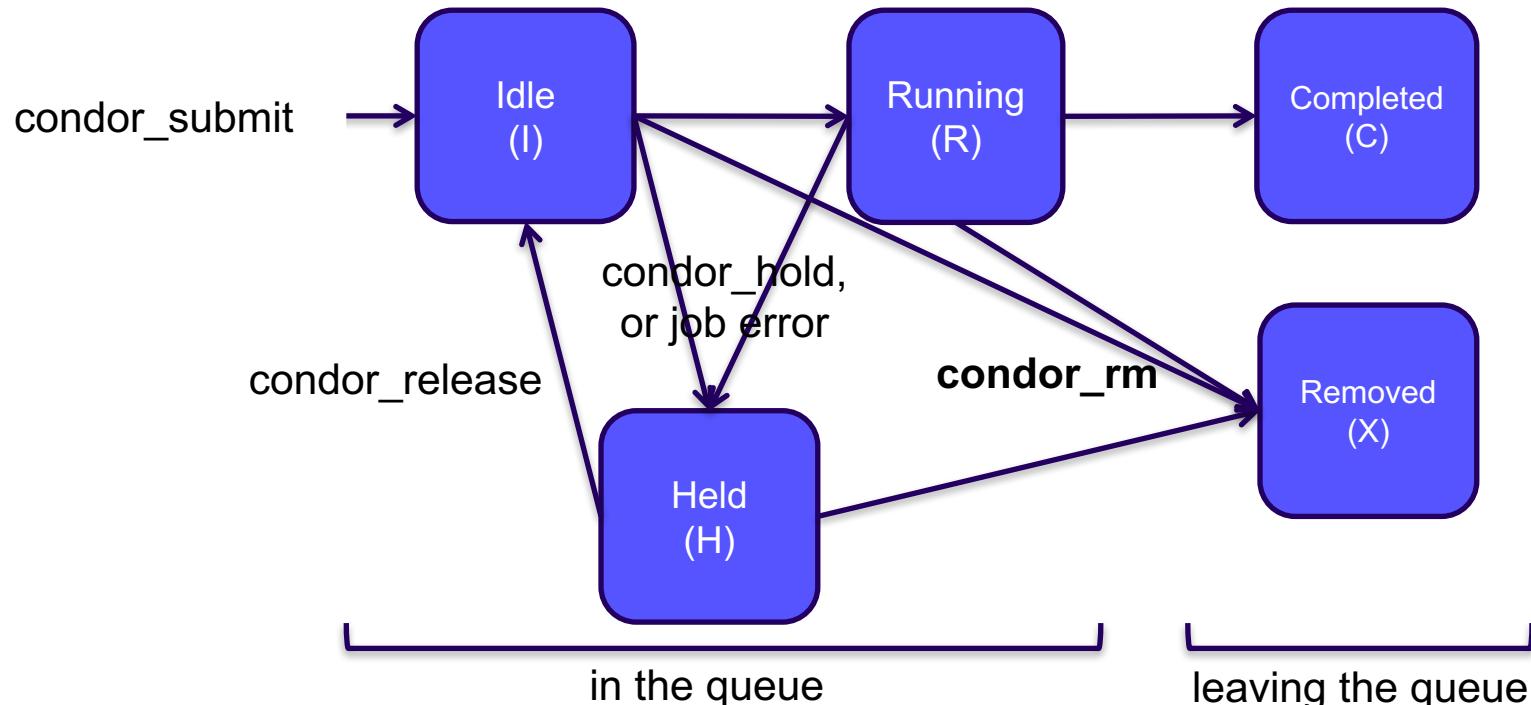
Job States, Revisited



Job States, Revisited



Job States, Revisited*



AUTOMATION AND OTHER FEATURES

Interactive Jobs

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

```
condor_submit -i submit_file
```

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

- Useful for testing and troubleshooting

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 an error code, leave it in the queue to run again. This is done via the automatic option `max_retries`.

```
max_retries = 5
```

More automation

- Check out the Intro to HTCondor talk from HTCondor Week 2017 for more on:
 - self-checkpointing
 - automatic hold/release (e.g. if job running too long)
 - auto-increasing memory request (e.g. if memory usage varies a lot across jobs)

Job Universes

- HTCondor has different “universes” for running specialized job types

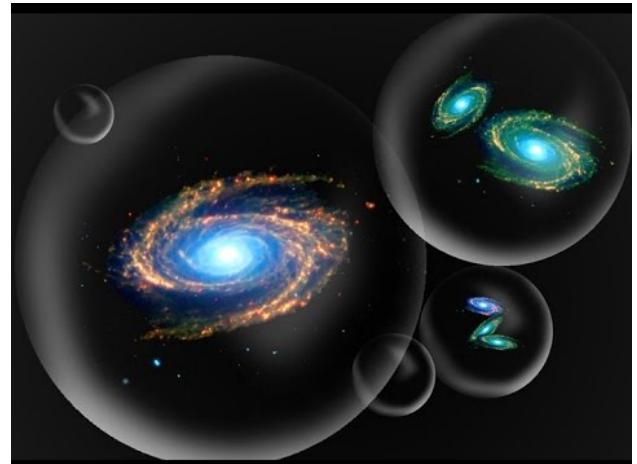
[HTCondor Manual: Choosing an HTCondor Universe](#)

- Vanilla (default)
 - good for most software

[HTCondor Manual: Vanilla Universe](#)

- Set in the submit file using:

```
universe = vanilla
```



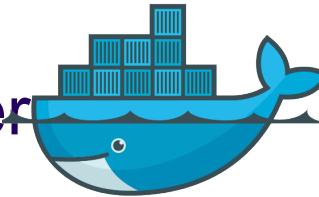
Other Universes

- Standard
 - Built for code (C, fortran) that can be statically compiled with `condor_compile`
- [HTCondor Manual: Standard Universe](#)
- Java
 - Built-in Java support
- [HTCondor Manual: Java Applications](#)
- Local
 - Run jobs on the submit node



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](#)
- Scheduler
 - Runs DAG workflows (next session)
- [HTCondor Manual: Parallel Applications](#)



Multi-CPU and GPU Computing

- Jobs that use multiple cores on a single computer can use the vanilla universe (parallel universe for multi-server MPI, where supported):

```
request_cpus = 16
```

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

```
request_gpus = 1
```

YOUR TURN!

Exercises!

- Ask questions!
- Lots of instructors around
- Coming up:
 - Now-12:15 Hands-on Exercises
 - 12:15 – 1:15 Lunch
 - 1:15 – 5:00 Afternoon sessions