An Introduction to Using HTCondor 2015

Karen Miller

The Team - 2014

Established in 1985,

to do research and development of distributed high-throughput computing

HT Stands for High Throughput

Throughput: the quantity of work done by an electronic computer in a given period of time (Dictionary.com)

HTCondor: a flexible batch queuing system

- Very configurable, adaptable
- Supports strong security methods
- Interoperates with many types of computing grids
- Manages both dedicated machines and non-dedicated machines (for cycle scavenging)
- Fault-tolerant: can survive crashes, network outages, any single point of failure

Your benefits

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

Parameter Sweep Example

Run the program once for each value of a variable.

1000 values is 1000 jobs

Each *job* needs resources when it runs:

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

Our scientist will be happy, since HTCondor will make the completion of the parameter sweep 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

Matchmaking
- associating a job with a machine resource

Central Manager
- central repository for the whole pool
- does matchmaking

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
  - I **require** a machine with MATLAB installed
- 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. No
- Rank (preferences):
  - This machine **prefers** to run Blast jobs

ClassAds

the language that HTCondor uses to represent information about:
jobs (job ClassAd),
machines (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")
```

String

Integer

Boolean

Boolean Expression

The Magic of Matchmaking

The matchmaker 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.

Some universes:

- vanilla
- vm
- grid
- java
- parallel
- standard

Vanilla Universe

- For many “serial” jobs
- 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

Any job can still use **stdin** (keyboard), **stdout** (screen) , and **stderr** , but files are used instead of the actual devices.
Specification is similar to Unix shell redirect:

```
$ ./myprogram <input.txt >output.txt
```

Make the Data Available

HTCondor will transfer files from the submit host *to* the execute host where the job runs. So, place these files in a place where HTCondor can access them.

HTCondor will also transfer result files *back from* the execute host *to* the submit host

3. Create a Submit Description File

- A plain ASCII text file
- File name extensions are irrelevant, although many use `.sub` or `.submit` as suffixes
- Describes 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 science.sub
# Lines beginning with a # are comments
# Note: the commands on the left are not
#   case sensitive, but file names
#   (on the right) are!
universe   = vanilla
executable = doscience
input      = data.in
output     = result.out
log        = doscience.log
queue
```

put 1 instance of the job in the queue

`input = infile`

Read job's standard input from **`infile`**

Like shell command: **`$ program < infile`**

`output = outfile`

Write job's standard output to **`outfile`**

Like shell command: **`$ program > outfile`**

`error = errorfile`

Write job's standard error to **`errorfile`**

Like shell command: **`$ program 2> errorfile`**

Log the Job's Activities

In the submit description file:

```
log = doscience.log
```

- Creates a log of job events, appended with all events as the job executes
- Good advice: *always* have a log file

Sample Portion of a 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 science.sub
Submitting job(s).
1 job(s) submitted to cluster 100.
```

`condor_submit` will

- parse the submit description file, checking for errors
- create a ClassAd that describes the job(s)
- place 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
ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
2.0      toni            3/14 12:01    0+00:03:48 R  0   0.0  env
3.0      hawking         3/14 12:48    0+00:00:00 H  0   0.0  script.sh
4.0      hawking         3/14 12:48    0+00:00:00 H  0   0.0  script.sh
.
.
.
98.0     bohr            3/14 15:59    0+00:00:00 I  0   0.0  atoms H
99.0     bohr            3/14 15:59    0+00:00:00 I  0   0.0  atoms H
100.0    chris           3/14 16:32    0+00:00:00 I  0   0.0  doscience

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. Generally used to limit the number files transferred.

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 execute host

= NO: rely on shared file system

= IF_NEEDED: transfer the files, if the submit and execute machine are not in the same file system domain (translation: use shared file system if available)

when_to_transfer_output

= ON_EXIT: transfer output files only when job completes

= ON_EXIT_OR_EVICT: transfer output files when job completes or is evicted

File Transfer Example

```
# changed science.sub file
universe                = vanilla
executable              = doscience
log                     = doscience.log
transfer_input_files    = extra.dat
transfer_output_files   = results.dat
should_transfer_files   = IF_NEEDED
when_to_transfer_output = ON_EXIT
queue
```

Command Line Arguments

```
universe   = vanilla
executable = doscience
arguments  = -c 299792458 -G 6.673e-112
. . .
queue
```

Invokes executable with

`doscience -c 299792458 -G 6.673e-112`

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

Job Id is

ClusterId.ProcId (ClassAd attributes)

- A set of related jobs is called a cluster
- Each cluster has a cluster number, an unsigned integer value unique to the job queue on a submit host
- Each individual job within a cluster is given a process number, 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, 2 Jobs

```
universe   = vanilla
executable = doscience

log        = doscience_0.log
input      = data_0.in
output     = result_0.out
queue                          job 102.0
log        = doscience_1.log
input      = data_1.in
output     = result_1.out
queue                          job 102.1
```

File Organization

Expand this to all 1000 jobs:

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

- 3 files × 1,000 jobs = 3,000 files
- The submit description file is 4,000+ lines

Too many files in 1 directory.

Better Organization

- Create a subdirectory for each job, intentionally named

  `run_0`, **`run_1`**, … **`run_999`**
- Implement the creation of directories with a program (such as Python or Perl)
- Create or place input files in each of these

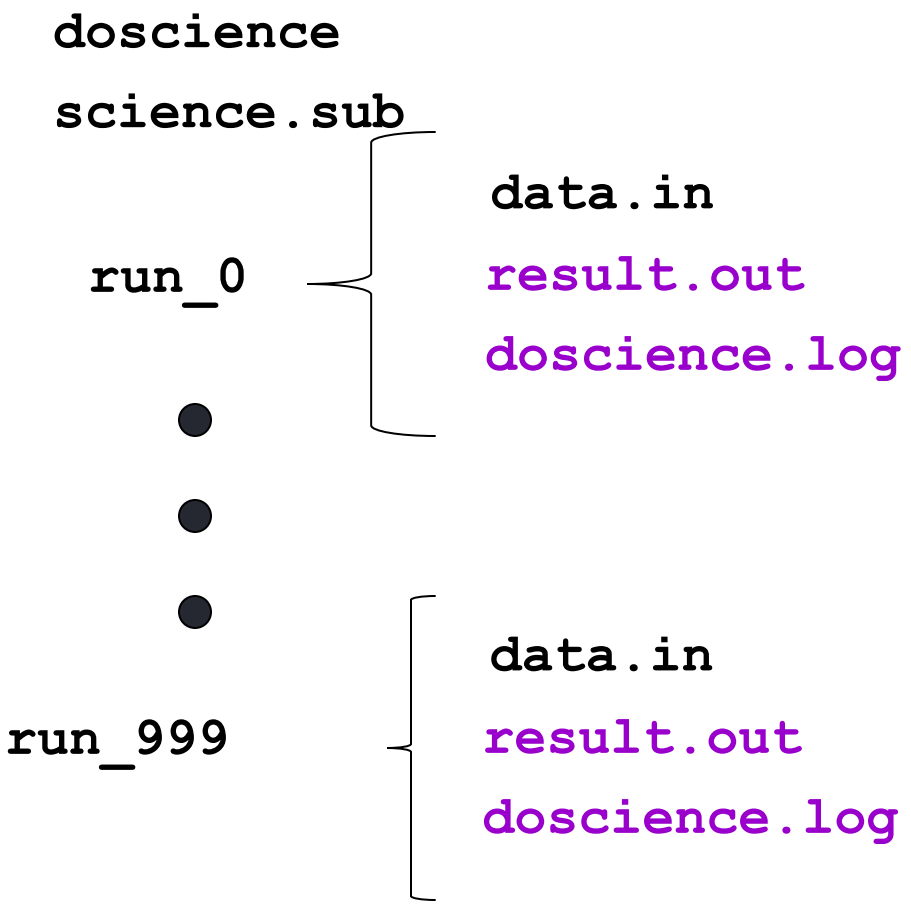
  `run_0/data.in`

  `run_1/data.in`

  …

  `run_999/data.in`
- The output and log files for each job will be created by the job, when the job runs.

directory structure and contents



Submitter or script creates black-font files

HTCondor creates purple-font files

Better Submit Description File

```
# Cluster of 1,000 jobs
universe     = vanilla
executable    = doscience
log           = doscience.log
output        = result.out
input         = data.in
initialdir    = run_0
queue                                  job 103.0
initialdir    = run_1
queue                                  job 103.1
```

This file contains 998 more instances of `initialdir` and `queue`.

WANTED: queue all instances of this job with the single command:

```
queue 1000
```

Submit Description File Macros

Within the submit description file, HTCondor supports automatic variables:

`$(Process)` will be expanded to be the same as the ClassAd attribute `ProcId` for each job in the cluster.

For this example, values will be 0 – 999 for the 1,000 jobs.

Using $(Process)

- Specify the initial directory for each job
  `initialdir = run_$(Process)`
  becomes
  `run_0`, `run_1`, `...`, `run_999`
- This automatic variable may be used other places within the submit description file. For example, specify command-line arguments
  `arguments = -n $(Process)`
  becomes
  `-n 0`, `-n 1`, `...`, `-n 999`

(Best) Submit Description File

```
# Example: one cluster of 1000 jobs

universe   = vanilla
executable = doscience
log        = doscience.log
input      = data.in
output     = result.out
initialdir = run_$(Process)
queue 1000
```

Patience required to submit large numbers of jobs

```
$ condor_submit science.sub
Submitting job(s)
   ..................................
   ..........................
Logging submit event(s)
   ..................................
   ..................................
   ..................................
   ......
1000 job(s) submitted to cluster 104.
```

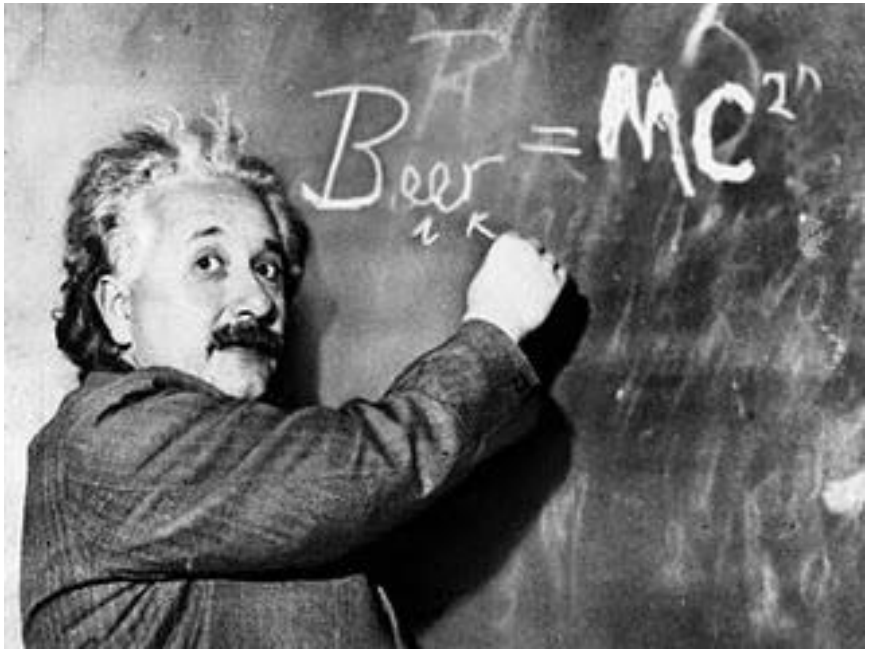
the Job Queue

```
$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:
   51883> : submit.chtc.wisc.edu
ID        OWNER  SUBMITTED   RUN_TIME  ST PRI SIZE CMD
104.1      chris 3/14 16:58 0+00:00:03 R 0 9.8 doscience
104.2      chris 3/14 16:58 0+00:00:01 I 0 9.8 doscience
104.3      chris 3/14 16:58 0+00:00:00 I 0 9.8 doscience
...
104.998    chris 3/14 16:58 0+00:00:00 I 0 9.8 doscience
104.999    chris 3/14 16:58 0+00:00:00 I 0 9.8 doscience

999 jobs; 998 idle, 1 running, 0 held
```

HTCondor watches over the jobs, runs each one to completion once, restarting any that do not finish.

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 !*

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 = (   \
   HasMathematicaInstalled =?= True )
queue
```

Specify Needed Resources

Automatically appended to job **`Requirements`**

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

`request_disk` – the amount of disk space (in KB) 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; a rank of 100 is higher than a rank of 6
- Like **`Requirements`**, is evaluated against attributes from machine ClassAds

```
universe     = vanilla
executable   = doscience
. . .
rank         = (KFLOPS*10000) + Memory
queue 1000
```

Job Policy Expressions

- Do not remove if exits with a signal:

```
on_exit_remove = ExitBySignal == False
```

- Place on hold if exits with nonzero status or ran for less than an hour:

```
on_exit_hold =
  ( (ExitBySignal==False) && (ExitSignal != 0) ) ||
  ( (ServerStartTime - JobStartDate) < 3600)
```

- Place on hold if job has spent more than 50% of its time suspended:

```
periodic_hold =
  ( CumulativeSuspensionTime >
    (RemoteWallClockTime / 2.0) )
```

~~Problems~~

Solutions

Jobs Are Idle

Our scientist runs **condor_q** and finds all jobs are idle:

```
$ condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510>
   :x.cs.wisc.edu
ID  OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
5.0 chris     3/14 12:23  0+00:00:00  I 0   9.8  doscience
5.1 chris     3/14 12:23  0+00:00:00  I 0   9.8  doscience
5.2 chris     3/14 12:23  0+00:00:00  I 0   9.8  doscience
5.3 chris     3/14 12:23  0+00:00:00  I 0   9.8  doscience
5.4 chris     3/14 12:23  0+00:00:00  I 0   9.8  doscience
5 jobs; 5 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 doscience.log
000 (005.000.000) 03/13 14:47:31 Job submitted from
  host: <128.105.121.53:510>
...
007 (005.000.000) 03/13 15:02:00 Shadow exception!
        Error from starter on gig1.cs.wisc.edu:
   Failed to open '/scratch.
  1/chris/workspace/test3/run_0/data.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
Name                OpSys     Arch   State     Activity LoadAv Mem   ActvtyTime
slot1@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  4599  0+00:10:13
slot2@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  1024  1+19:10:36
slot3@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     0.990  1024  1+22:42:20
slot4@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  1024  0+03:22:10
slot5@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  1024  0+03:17:00
slot6@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  1024  0+03:09:14
slot7@c002.chtc.wi  LINUX     X86_64 Claimed   Busy     1.000  1024  0+19:13:49
...
slot7@exec-2.chtc.  WINDOWS     INTEL  Owner   Idle     0.000   511  0+00:24:17
slot8@exec-2.chtc.  WINDOWS     INTEL  Owner   Idle     0.030   511  0+00:45:01

                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        INTEL/WINDOWS   104    78      16        10       0          0        0
         X86_64/LINUX   759   170     587         0       0          1        0

                Total   863   248     603        10       0          1        0
```

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
```

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.
---
102.001: 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
```

(continued)

```
The Requirements expression for your job is:
( TARGET.Arch == "X86_64" ) &&
( TARGET.OpSys == "WINDOWS" ) &&
( TARGET.Disk >= RequestDisk ) &&
( TARGET.Memory >= RequestMemory ) &&
( TARGET.HasFileTransfer )

Suggestions:
   Condition                Machines Matched    Suggestion
   ---------                ----------------    ----------
1  ( TARGET.OpSys == "WINDOWS" ) 0   MODIFY TO "LINUX"
2  ( TARGET.Arch == "X86_64" )   3137
3  ( TARGET.Disk >= 1 )          3184
4  ( TARGET.Memory >= ifthenelse(MemoryUsage isnt
undefined,MemoryUsage,1) )       3184
5  ( TARGET.HasFileTransfer )    3184
```

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
```

Lots of Short-Running Jobs

Starting a job is somewhat expensive, in terms of time (overhead). 2 items that might help:

1. Batch short jobs together

   - write a wrapper script that will run a set of the jobs in series

   - the wrapper script becomes the job executable

2. There are some configuration variables that may be able to help. Contact a staff person for more info.

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  chris 4/18 06:52 1+12:10:05 R 0    10.0 doscience

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
. . .
```

After this tutorial, here are some places you might find help:

1. HTCondor manual
2. htcondor-users mailing list
   `https://lists.cs.wisc.edu/mailmain/listinfo/htcondor-user`
3. wiki
   `https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki`
4. developers

Other universes are better than

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

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*.

Standard Universe Features

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

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

```
# 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
+ParallelShutdownPolicy = "WAIT_FOR_ALL"
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 *HOWTO Recipes for configuration, fancy tricks,*
"How to allow some jobs to claim the whole machine instead of one slot"

VM Universe

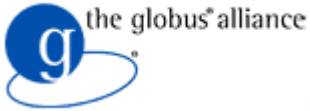
- A virtual machine instance is the HTCondor job
- The vm universe offers
  - job sandboxing
  - checkpoint and migration
  - safe elevation of privileges
  - cross-platform submission
- HTCondor supports VMware, Xen, and KVM
- Input files can be imported as CD-ROM image
- When the VM shuts down, the 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
  - GCE (Google Compute Engine)
  - BOINC

LSF®


Java Universe

More than

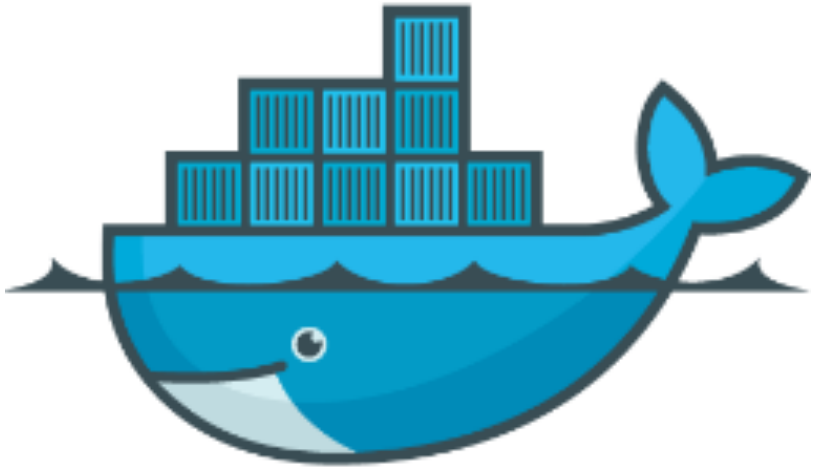
```
$  java mysimulator
```

- Knows which machines have a JVM installed
- Knows the location, version, and performance of the 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.

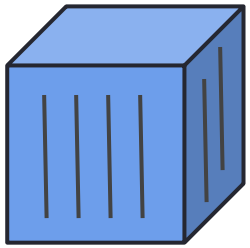
Java Universe Example

```
# 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
```

Docker Universe

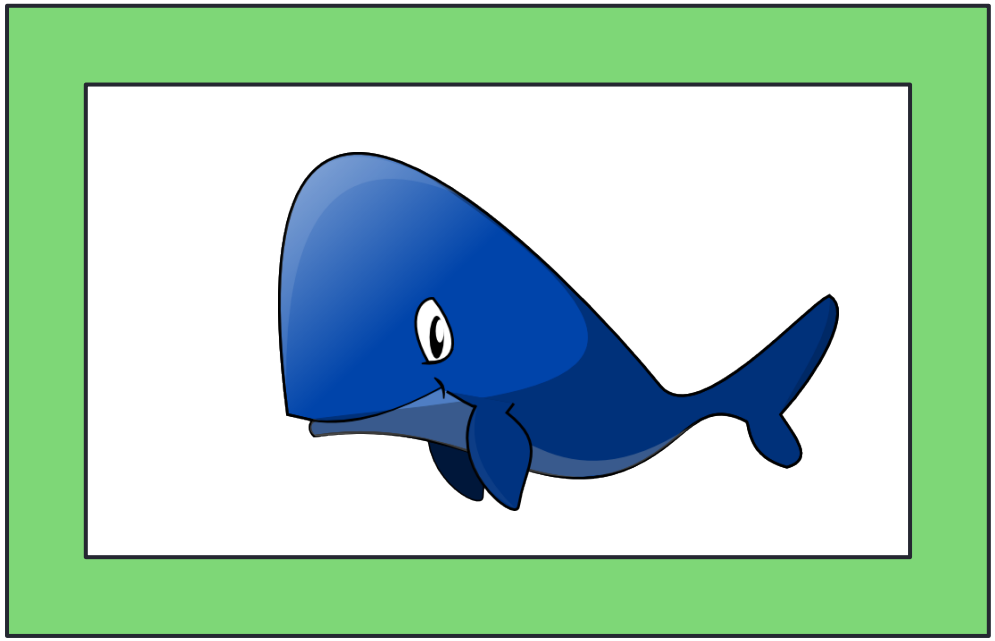


Docker Universe

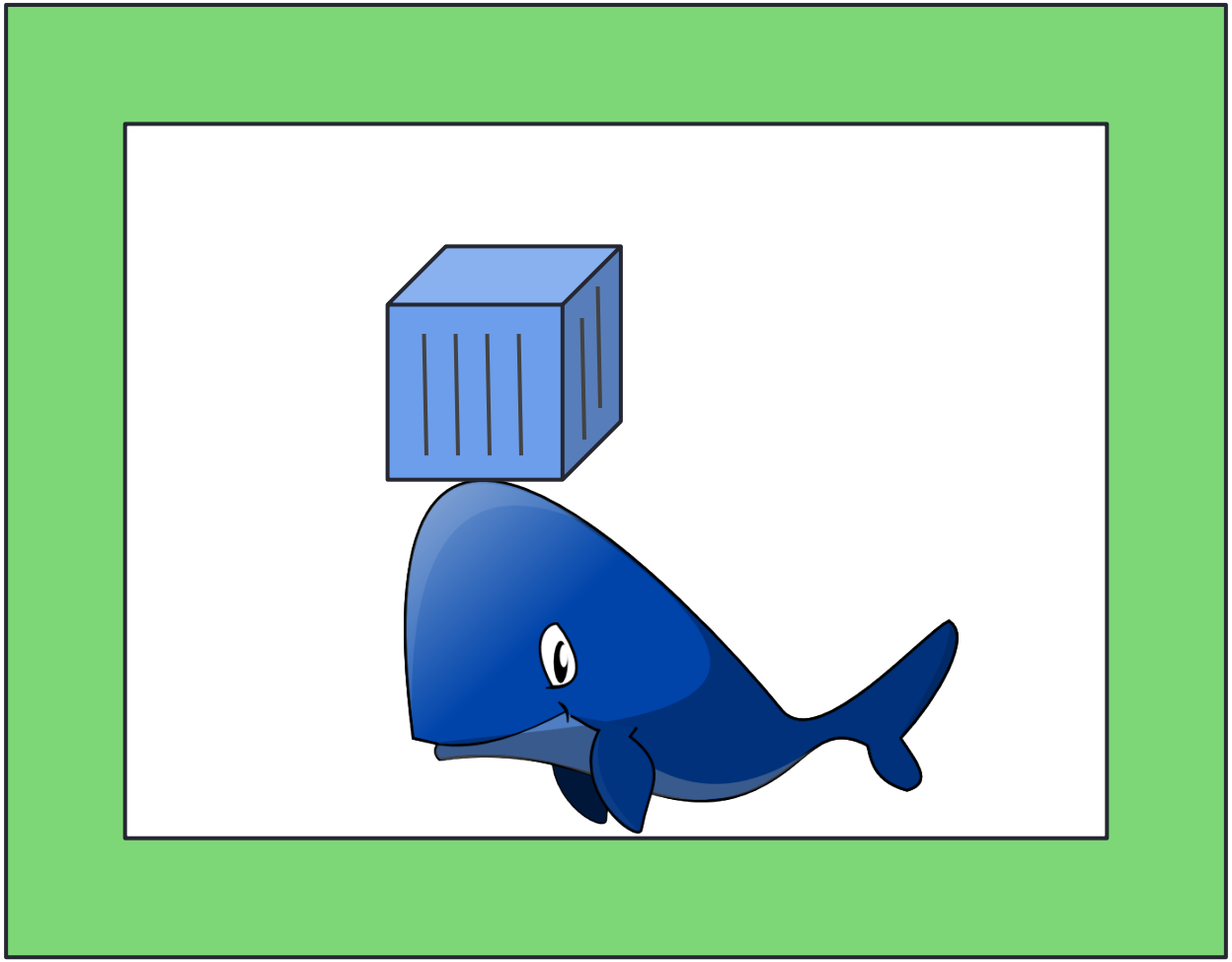


the container is an HTCondor job

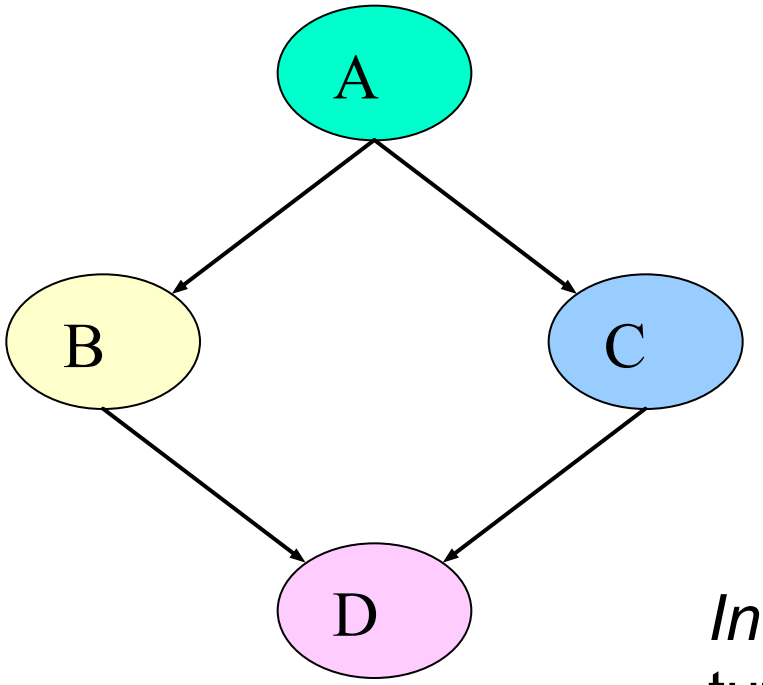
Docker-capable execute host



Docker Universe



Advanced Features



DAGMan specifies dependencies between jobs that can be described by a DAG.

Interested? Attend Kent's tutorial on managing workflows with DAGMan.

Wanted:

Queue one job for each item in a list

```
queue 1 input in A.dat, B.dat, C.dat
```

list of 3 items

Results in

```
input = A.dat
queue
input = B.dat
queue
input = C.dat
queue
```

Wanted:

Queue one job for each file within a directory.

```
input = $(filename)
queue filename matching files dataset1/*
```

Results in

```
input = dataset1/A.dat
queue
input = dataset1/B.dat
queue
input = dataset1/test35.dat
queue
. . .
```

FILE GLOBBING

In Review

With HTCondor's help, both you and scientist Chris 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

Email as Feedback

- HTCondor may send 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

InitialDir

- Identifies a directory for file input and output.
- Also provides a directory (on the **submit** machine) for the job log, when a full path is not specified.

```
# 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

$$(<attribute>) will be replaced by the value of the specified attribute from the machine ClassAd

Example:

Machine ClassAd has:

```
CosmosData = "/local/cosmos/data"
```

Submit description file has

```
Executable   = cosmos
Requirements = (CosmosData =!= UNDEFINED)
Arguments    = -d $$(CosmosData)
```

Results in the job invocation:

```
cosmos -d /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)
  - Windows, many versions
- Repositories
  - YUM: RHEL 4, 5, and 6

    ```
    $ yum install condor.x86_64
    ```
  - APT: Debian 6 and 7

    ```
    $ apt-get install condor
    ```

HTCondor Releases

- Stable and Developer Releases
  - Version numbering scheme similar to that of the (pre 2.6) Linux kernels …
- Numbering: major.minor.release
  - If minor is even (a.b.c): **Stable** series
    - Very stable, mostly bug fixes
    - Current: 8.0
  - If minor is odd (a.b.c): **Developer** series
    - New features, may have some bugs
    - Current: 8.1

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 taking a checkpoint
condor_compile	Link HTCondor library with job