Using HTCondor

European HTCondor Site Admins Meeting CERN December 2014

Job

- HTCondor's quanta of work, like a Unix process
- Has many attributes
  - Executable, input files, output files
  - State such as current working directory
  - Owner
- Can be an element of a larger batch processing workflow

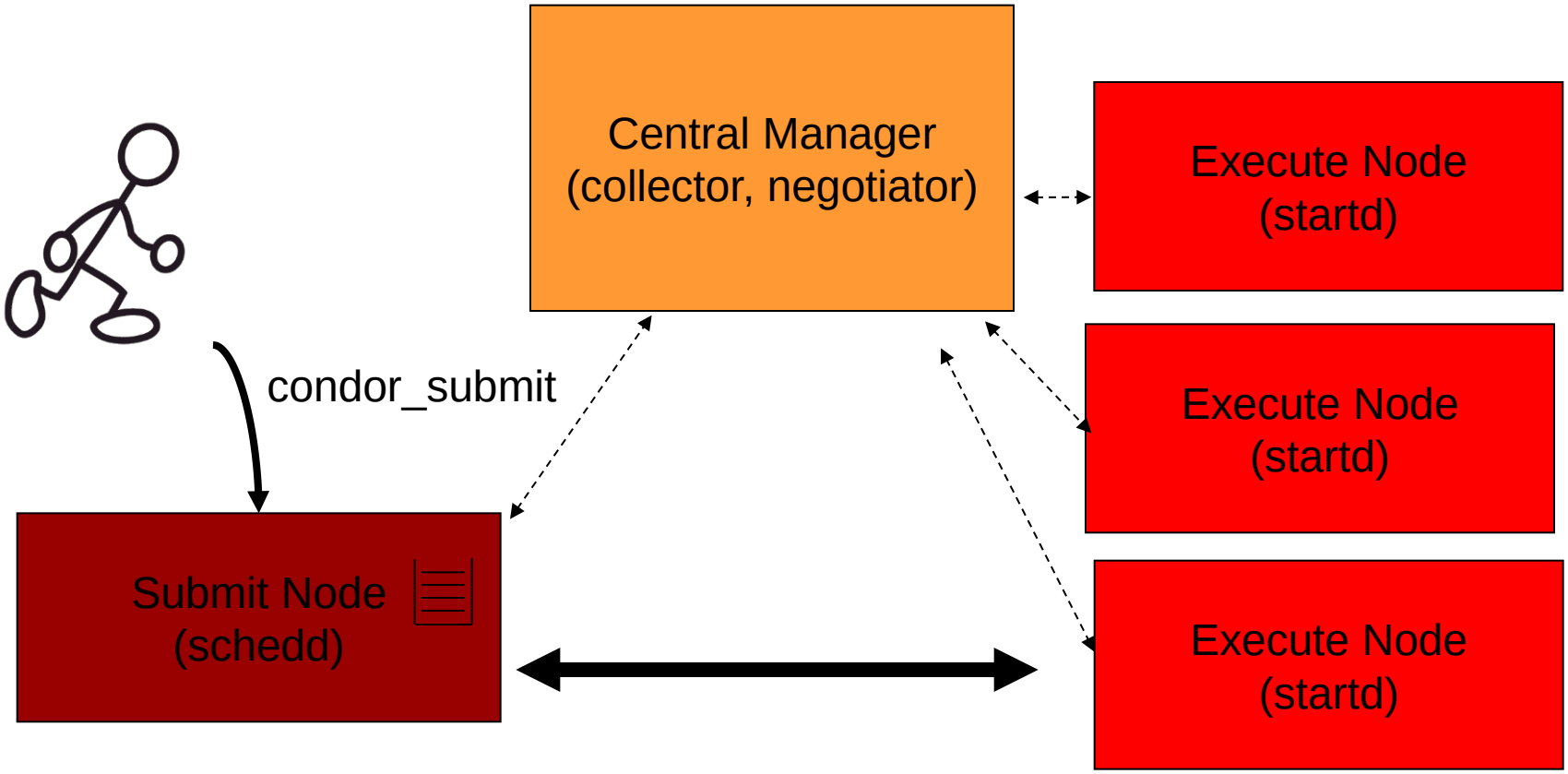
Jobs Have Wants & Needs

- Jobs state their requirements and preferences, and attributes about themselves:
  - Requirements:
    - I **require** a Linux/x86 platform
    - I **require** 500MB RAM
  - Preferences ("Rank"):
    - I **prefer** a machine in the chemistry department
    - I **prefer** a machine with the fastest floating point
  - Custom Attributes:
    - I am a job of type “analysis”

Machines Do Too!

- Machines specify:
  - Requirements:
    - **Require** that jobs run only when there is no keyboard activity
    - **Never** run jobs labeled as “production”
  - Preferences ("Rank"):
    - I **prefer** to run Todd’s jobs
  - Custom Attributes:
    - I am a machine in the chemistry department

HTCondor brings them together



ClassAds: The *lingua franca* of HTCondor

What are ClassAds?

ClassAds is a language for objects (jobs and machines) to

- Express attributes about themselves
- Express what they require/desire in a “match” (similar to personal classified ads)

Structure : Set of attribute name/value pairs, where the value can be a literal or an expression. Semi-structured, no fixed schema.

Example

Pet Ad

```
Type  = "Dog"
Requirements =
    DogLover =?= True
Color = "Brown"
Price = 75
Sex = "Male"
AgeWeeks = 8
Breed = "Saint Bernard"
Size = "Very Large"
Weight = 27
```

Buyer Ad

```
AcctBalance  = 100
DogLover = True
Requirements =
  (Type == "Dog")   &&
  (TARGET.Price <=
   MY.AcctBalance) &&
  ( Size == "Large" ||
    Size == "Very Large" )
Rank =
 100* (Breed == "Saint
 Bernard") - Price
. . .
```

ClassAd Values

- Literals
  - Strings (“RedHat6”), integers, floats, boolean (true/false), …
- Expressions
  - Similar look to C/C++ or Java : operators, references, functions
  - References: to other attributes in the same ad, or attributes in an ad that is a candidate for a match
  - Operators: +, -, *, /, <, <=,>, >=, ==, !=, &&, and || all work as expected
  - Built-in Functions: if/then/else, string manipulation, regular expression pattern matching, list operations, dates, randomization, math (ceil, floor, quantize,…), time functions, eval, …

Four-valued logic

- ClassAd Boolean expressions can return four values:
  - True
  - False
  - Undefined (a reference can't be found)
  - Error (Can't be evaluated)
- Undefined enables explicit policy statements *in the absence of data* (common across administrative domains)
- Special meta-equals (=?=) and meta-not-equals (=!=) will never return Undefined

```
[
  HasBeer = True
  GoodPub1 = HasBeer == True
  GoodPub2 = HasBeer =?= True
]
```

```
[
  GoodPub1 = HasBeer == True
  GoodPub2 = HasBeer =?= True
]
```

ClassAd Types

- HTCondor has many types of ClassAds
  - A "Job Ad" represents a job to Condor
  - A "Machine Ad" represents a computing resource
  - Others types of ads represent other instances of other services (daemons), users, accounting records.

The Magic of Matchmaking

- Two ClassAds can be matched via special attributes: Requirements and Rank
- Two ads match if both their Requirements expressions evaluate to True
- Rank evaluates to a float where higher is preferred; specifies the which match is desired if several ads meet the Requirements.
- Scoping of attribute references when matching
  - MY.name – Value for attribute “name” in local ClassAd
  - TARGET.name – Value for attribute “name” in match candidate ClassAd
  - Name – Looks for “name” in the local ClassAd, then the candidate ClassAd

Example

Pet Ad

```
Type  = "Dog"
Requirements =
    DogLover =?= True
Color = "Brown"
Price = 75
Sex = "Male"
AgeWeeks = 8
Breed = "Saint Bernard"
Size = "Very Large"
Weight = 27
```

Buyer Ad

```
AcctBalance  = 100
DogLover = True
Requirements =
  (Type == "Dog")   &&
  (TARGET.Price <=
   MY.AcctBalance) &&
  ( Size == "Large" ||
    Size == "Very Large" )
Rank =
 100* (Breed == "Saint
 Bernard") - Price
. . .
```

Getting Started: Submitting Jobs to HTCondor

- Get access to submit host
- Choose a "Universe" for your job
- Make your job "batch-ready"
  - Includes making your data available to your job
- Create a *submit description* file
- Run *condor_submit* to put your job(s) in the queue
- Relax while Condor manages and watches over your job(s)

Choose the job "Universe"

- Controls how HTCondor handles jobs
- HTCondor's many universes include:
  - Vanilla (aka regular serial job)
  - Local
  - Parallel
  - Grid
  - Java
  - VM
  - Standard

Hello World Submit File

```
# Simple condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
Executable = cosmos
Output     = cosmos.out
Input      = cosmos.in
Queue 1
```

- Job's executable
- Job's STDOUT
- Job's STDIN
- Put the job in the queue!

condor_submit & condor_q

```
% condor_submit sim.submit
Submitting job(s).
1 job(s) submitted to cluster 1.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda          6/16 06:52   0+00:00:00 I  0    0.0 sim.exe

1 jobs; 1 idle, 0 running, 0 held

%
```

View the full ClassAd

```
% condor_q -long

-- Submitter: perdita.cs.wisc.edu :
   <128.105.165.34:1027> :
MyType = “Job”
TargetType = “Machine”
ClusterId = 1
QDate = 1150921369
CompletionDate = 0
Owner = “frieda”
RemoteWallClockTime = 0.000000
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
RemoteUserCpu = 0.000000
RemoteSysCpu = 0.000000
ExitStatus = 0
...
```

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**

Submit file with multiple procs

```
Universe                = vanilla
Executable              = cosmos
Log                     = cosmos.log
Output                  = cosmos.out
Input                   = cosmos.in
InitialDir = run_0
Queue 1                 # Job 103.0 (Cluster 103, Process 0)
InitialDir = run_1
Queue 1                 # Job 103.1 (Cluster 103, Process 1)
```

-
-
-

Some submit file macros

`$(Process)` will be expanded to the process number for each job in the cluster; corresponds to ProcID attribute

$$(X) will be expanded to the value of the attribute named X in the matched machine classad

$$([classad expression]) will be expanded to the value of expression evaluated in the context of job and matched machine classad

Using $(Process)

- The initial directory for each job can be specified
  `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`

Specify Needed Resources

Items appended to job ad **`Requirements by condor_submit:`**

`request_cpus` – the number of CPUs (cores) that the job needs.

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

`re`

`re`

condor_submit handles "request_cpus = 2" by placing the following into the job classad:

RequestCpus = 2

Requirements = TARGET.Cpus >= RequestCpus

Logging your Job's Activities

- Create a log of job events
- Add to submit description file:

```
log = cosmos.log
```

- The Life Story of a Job
  - Shows all events in the life of a job
  - Helps users figure out what happened if results are not what they expect
  - Libraries to parse them provided

Sample HTCondor Job Event 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)
...
```

Another Example

```
Executable = cosmos
Output = cosmos.out
Log = cosmos.err
# Do each run in its own Subdirectory
Initialdir  = Run_$(Process)
# Need at least 1 CPU, 1 GPU, and 2GB RAM
request_cpus = 1
request_gpus = 1
request_memory = 2GB
# Run 1000 different data sets
Queue 1000
```

Another Example

```
Executable = cosmos
Output = cosmos.out
Log = cosmos.err
# Do each run in its own Subdirectory
Initialdir  = Run_$(Process)
# Need at least 1 CPU, 1 GPU, and 8GB RAM
request_cpus = 1
request_gpus = 1
request_memory = 8GB




# Run 1000 different data sets
Queue 1000
```

Bells and Whistles Example

```
Executable = cosmos
Output = cosmos.out
Log = cosmos.err
# Do each run in its own Subdirectory
Initialdir  = Run_$(Process)
# Need at least 1 CPU, 1 GPU, and 8GB RAM
request_cpus = 1
request_gpus = 1
request_memory = 8GB
# Location of 'cosmos_data' is in machine ad
Arguments = -data $$(cosmos_data) -ram $$([TARGET.Memory*.9])
# Require Red Hat 6 machines, desire fast ones
Requirements = OpSysAndVer =?= "RedHat6" &&
    cosmos_data =!= UNDEFINED
Rank = 100000*KFlops + Memory
# Label as an analysis job
+JobType = "analysis"
# Run 1000 different data sets
Queue 1000
```

Some common problems with jobs your users may encounter…

Jobs Are Idle

User runs **condor_q** and finds all his 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 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.1 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.2 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.3 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.4 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.5 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.6 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
5.7 einstein  4/20 12:23  0+00:00:00  I 0   9.8  cosmos
8 jobs; 8 idle, 0 running, 0 held
```

Exercise a little patience

- On a large/busy pool, it could take a few minutes to match a machine to the first job when a new batch of jobs is submitted.

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
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
```

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
```

(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
```

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*, …
  - Ssh session started alongside running job, with same environment, uid, credentials, etc.
  - Still managed by scheduling policies

Interactive Debug Example

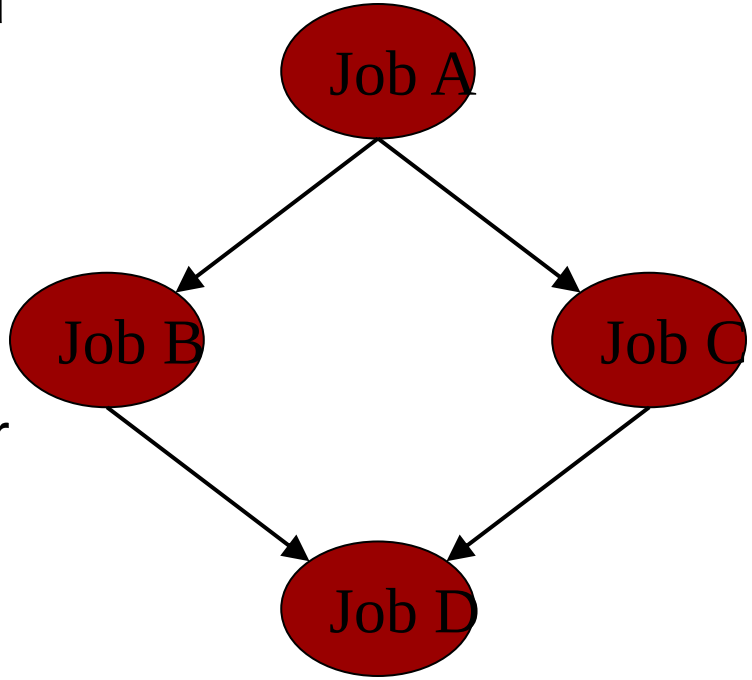
```
$ condor_q
-- Submitter: cosmos.phy.wisc.edu : <128.105.165.34:1027>

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
. . .
```

DAGMan

- **Directed Acyclic Graph Manager**
- DAGMan allows you to specify the *dependencies* between your HTCondor jobs, so it can *manage* them automatically for you.
- (e.g., “Don’t run job “B” until job “A” has completed successfully.”)

What is a DAG?

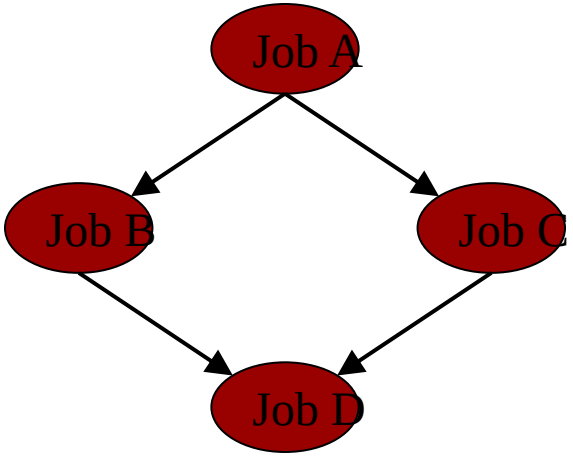
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as long as there are no loops!



Defining a DAG

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

```
# diamond.dag
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
```



- each node will run the HTCondor job specified by its accompanying HTCondor submit file

Submitting a DAG

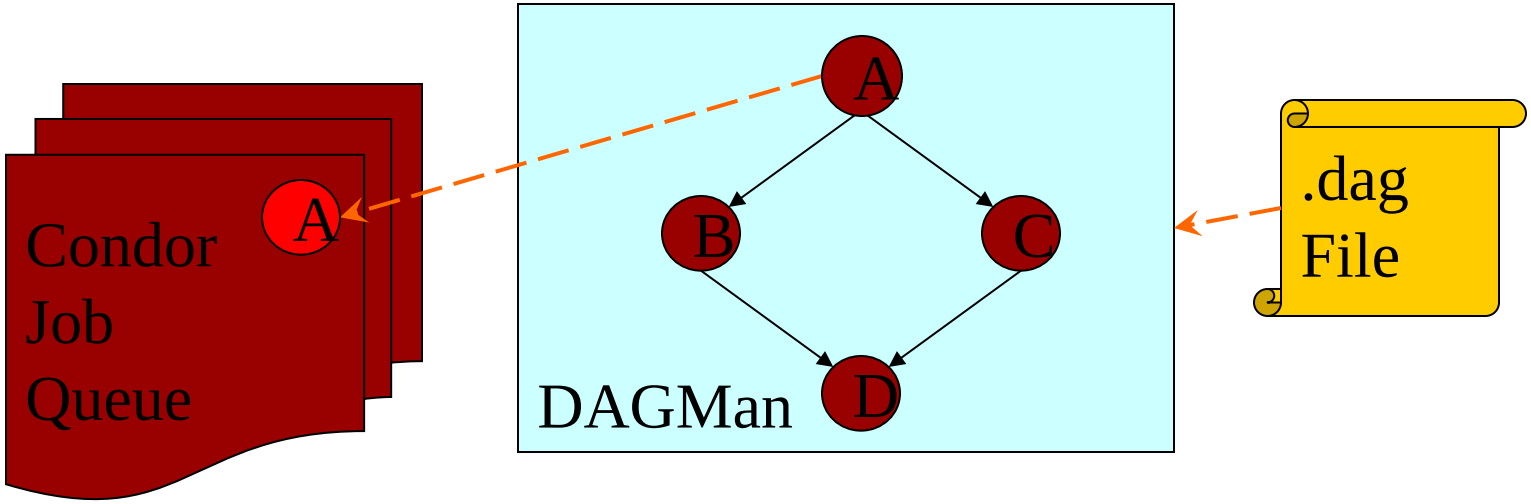
- To start your DAG, just run ***condor_submit_dag*** with your .dag file:

```
% 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 a Condor job, so you don't have to baby-sit it.

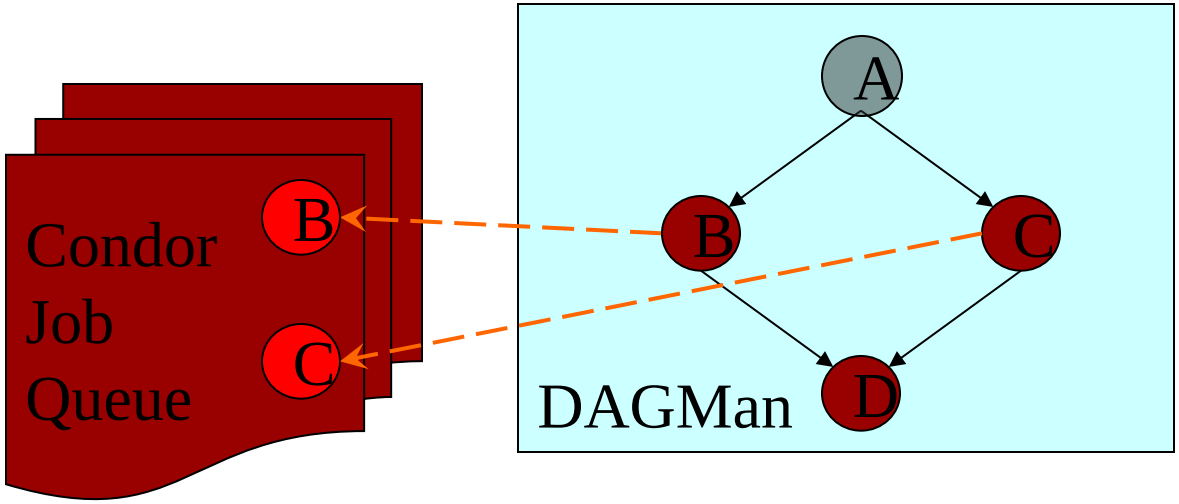
Running a DAG

- DAGMan acts as a “meta-scheduler”, managing the submission of your jobs to Condor based on the DAG dependencies.



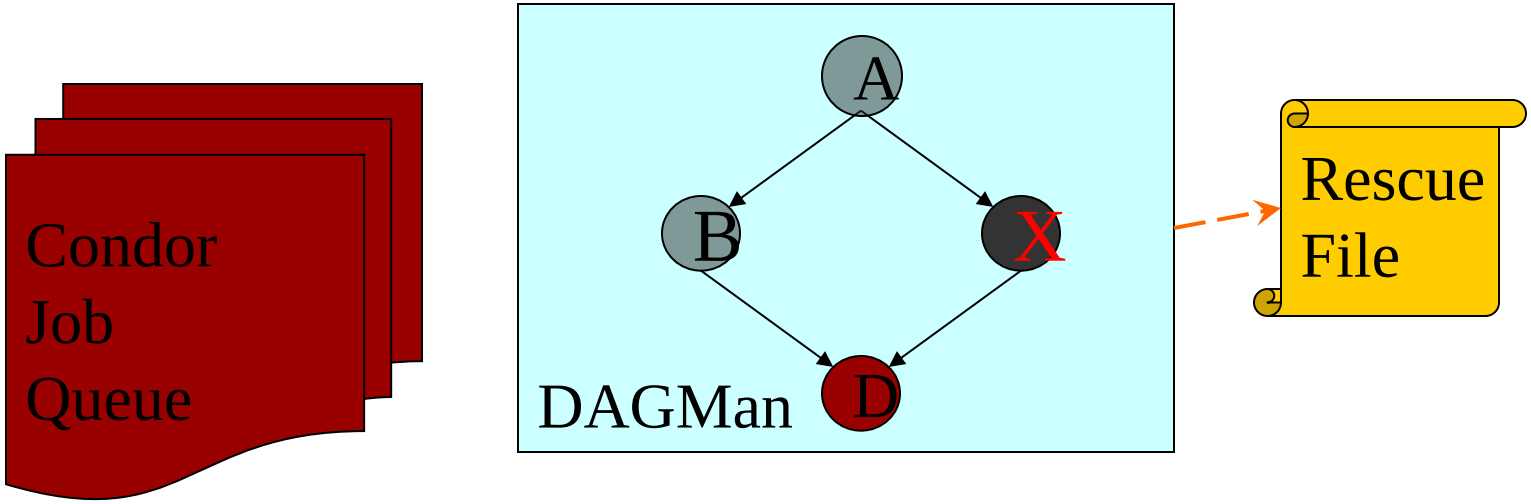
Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.



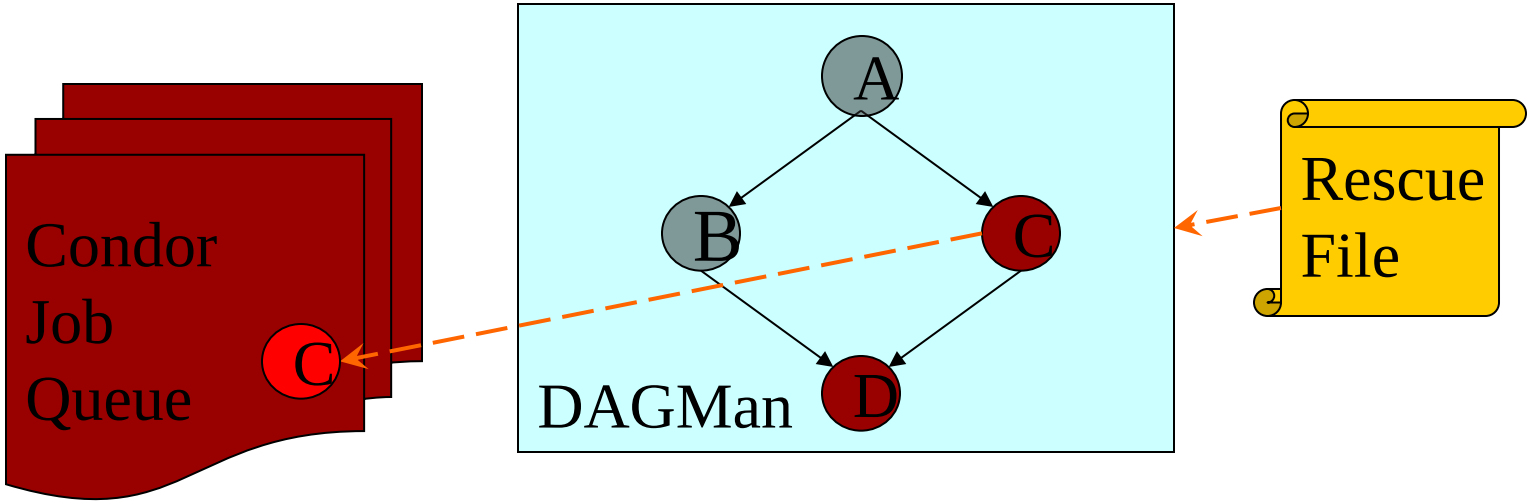
Running a DAG (cont'd)

- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *"rescue" file* with the current state of the DAG.



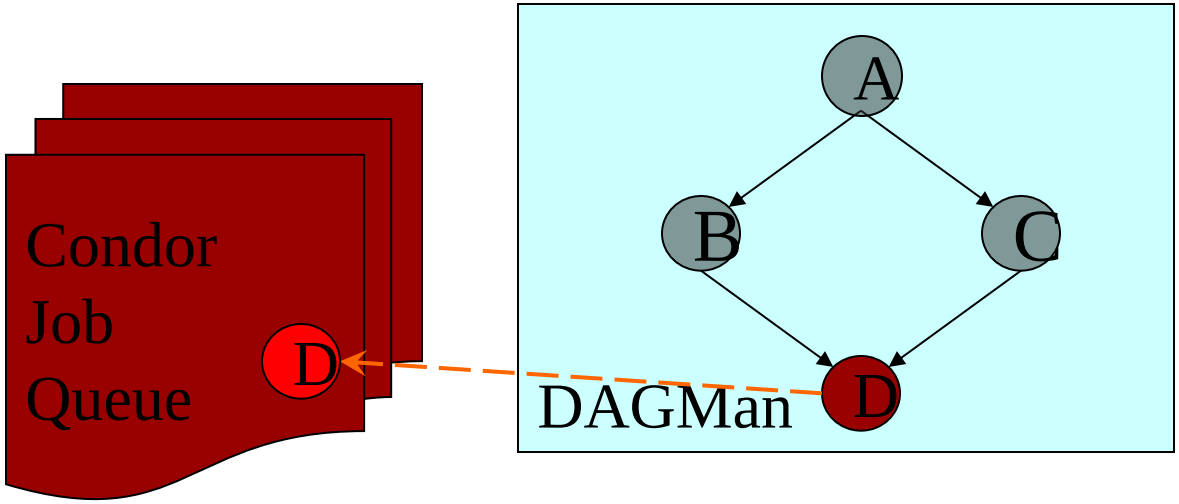
Recovering a DAG

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



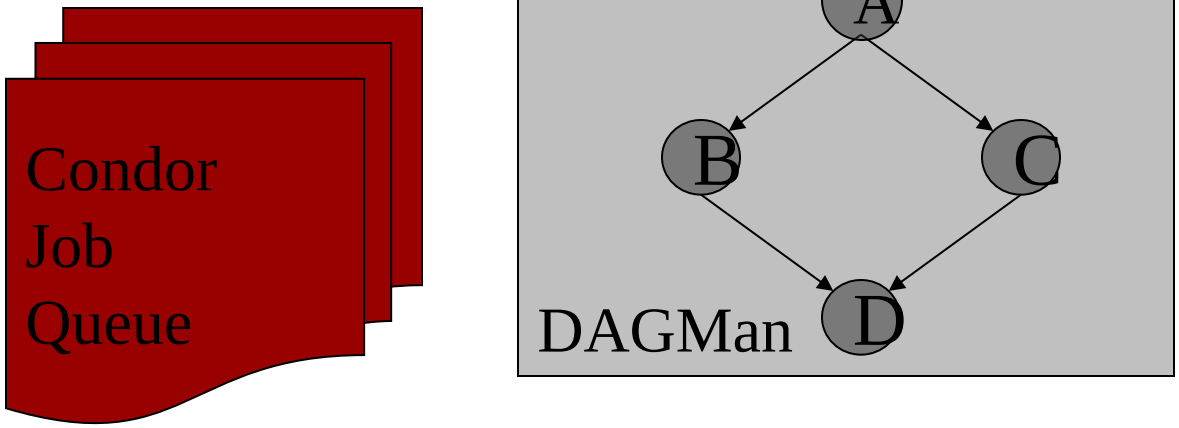
Recovering a DAG (cont'd)

- Once that job completes, DAGMan will continue the DAG as if the failure never happened.

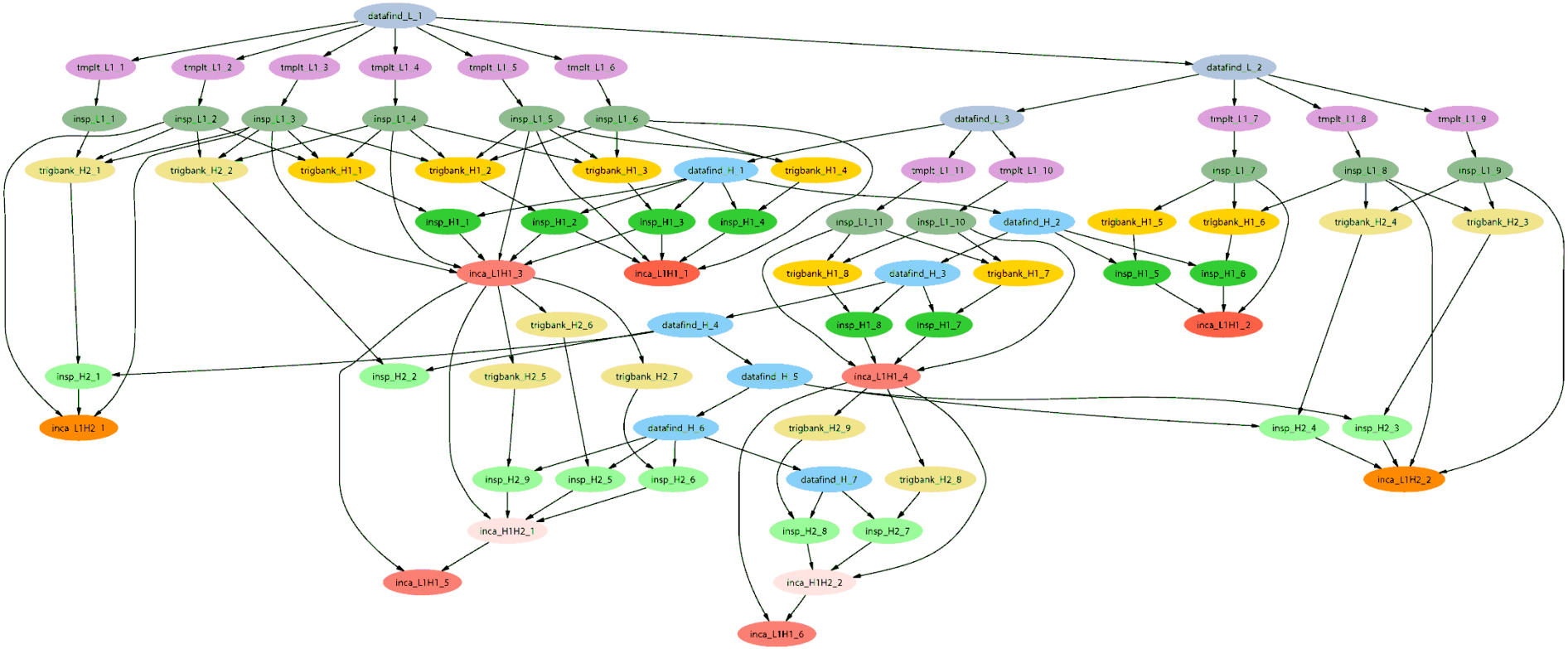


Finishing a DAG

- Once the DAG is complete, the DAGMan job itself is finished, and exits.



LIGO inspiral search application



Inspiral workflow application is the work of Duncan Brown, Caltech, Scott Koranda, UW Milwaukee, and the LSC Inspiral group

Additional DAGMan Mechanisms

- Provides other handy features for workflow management…
  - nodes can have PRE & POST scripts
  - failed nodes can be automatically re-tried a configurable number of times
  - job submissions can be "throttled" in many different ways
  - nodes in DAG graphs can have priorities and traversal schemes (depth first, breadth first, …)
  - DAGs can be nested and spliced

General User Commands

condor_submit	Submit new Jobs
condor_status	View Pool Status
condor_q	View Job Queue
condor_q -analyze	Why job/machines fail to match?
condor_ssh_to_job	Create ssh session to active job
condor_submit -i	Submit interactive job
condor_hold / release	Hold a job, or release a held job
condor_run	Submit and block
condor_rm	Remove Jobs
condor_prio	Intra-User Job Prios
condor_history	Completed Job Info
condor_submit_dag	Submit new DAG workflow
condor_chirp	Access files/ad from active job
condor_compile	Link job with checkpoint library

Questions?

Thank You!

Condor File Transfer

HTCondor can transfer files between submit and execute nodes (eliminating the need for a shared filesystem) if desired:

- **ShouldTransferFiles**
  - **YES:** Always transfer files to execution site
  - **NO:** Always rely on a shared filesystem
  - **IF_NEEDED:** Condor will automatically transfer the files if the submit and execute machine are not in the same **FileSystemDomain** (Use shared file system if available)
- **When_To_Transfer_Output**
  - **ON_EXIT:** Transfer the job's output files back to the submitting machine only when the job completes
  - **ON_EXIT_OR_EVICT:** Like above, but also when the job is evicted

Condor File Transfer, cont

- `Transfer_Input_Files`
  - List of files that you want Condor to transfer to the execute machine
- `Transfer_Output_Files`
  - List of files that you want Condor to transfer from the execute machine
  - If not specified, Condor will transfer back all new or modified files in the execute directory

Simple File Transfer Example

```
# Example submit file using file transfer
Universe                    = vanilla
Executable                  = cosmos
Log                         = cosmos.log
ShouldTransferFiles         = YES
Transfer_input_files        = cosmos.dat
Transfer_output_files       = results.dat
When_To_Transfer_Output     = ON_EXIT
Queue
```