Workflows with HTCondor’s DAGMan

Monday PM, Lecture 1
Lauren Michael
Questions so far?
Goals for this Session

• Why Create a Workflow?
• Describing workflows as *directed acyclic graphs* (DAGs)
• Workflow execution via DAGMan (DAG Manager)
WHY WORKFLOWS?
WHY DAGS?
Automation!

- **Objective:** Submit jobs in a particular order, automatically.

- Especially if: Need to replicate the same workflow multiple times in the future.
DAG = “directed acyclic graph”

- topological ordering of vertices (“nodes”) is established by directional connections (“edges”)
- “acyclic” aspect requires a start and end, with no looped repetition
  - can contain cyclic subcomponents, covered in later slides for DAG workflows

[Image of DAG diagram]

Wikipedia Commons

wikipedia.org/wiki/Directed_acyclic_graph
DESCRIBING WORKFLOWS WITH DAGMAN
2.10 DAGMan Applications
  • 2.10.1 DAGMan Terminology
  • 2.10.2 The DAG Input File: Basic Commands
  • 2.10.3 Command Order
  • 2.10.4 Node Job Submit File Contents
  • 2.10.5 DAG Submission
  • 2.10.6 File Paths in DAGs
  • 2.10.7 DAG Monitoring and DAG Removal
  • 2.10.8 Suspending a Running DAG
  • 2.10.9 Advanced Features of DAGMan
  • 2.10.10 The Rescue DAG
  • 2.10.11 DAG Recovery
  • 2.10.12 Visualizing DAGs with dot
  • 2.10.13 Capturing the Status of Nodes in a File
  • 2.10.14 A Machine-Readable Event History, the jobstate.log File
  • 2.10.15 Status Information for the DAG in a ClassAd
  • 2.10.16 Utilizing the Power of DAGMan for Large Numbers of Jobs
  • 2.10.17 Workflow Metrics
  • 2.10.18 DAGMan and Accounting Groups
An Example HTC Workflow

- User must communicate the “nodes” and directional “edges” of the DAG
Simple Example for this Tutorial

- The DAG input file will communicate the “nodes” and directional “edges” of the DAG
The DAG input file will communicate the "nodes" and directional "edges" of the DAG.

Look for links on future slides.

OSG Summer School 2017

HTCondor Manual: DAGMan Applications > DAG Input File
Basic DAG input file: JOB nodes, PARENT-CHILD edges

my.dag

```plaintext
JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```

- Node names are used by various DAG features to modify their execution by DAG Manager.
Basic DAG input file: **JOB** nodes, **PARENT-CHILD** edges

- Node names and filenames can be anything.
- Node name and submit filename do not have to match.

```
(my.dag)

JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C

(dag_dir)/
A.sub    B1.sub
B2.sub   B3.sub
C.sub    my.dag
(other job files)
```
Endless Workflow Possibilities

OSG Summer School 2017

https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
Endless Workflow Possibilities
Repeating DAG Components!!
DAGs are also useful for non-sequential work.

‘bag’ of HTC jobs

B1  B2  B3  …  BN

disjointed workflows

A  B

C  D  E

F  G  H

I

A  B

C  D  E

F  G  H

I
Basic DAG input file:
**JOB** nodes, **PARENT-CHILD** edges

```
my.dag

JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```
SUBMITTING AND MONITORING A DAGMAN WORKFLOW
Submitting a DAG to the queue

• Submission command:

   `condor_submit_dag dag_file`

   

   `condor_submit_dag my.dag`

   

   File for submitting this DAG to HTCondor : mydag.dag.condor.sub
   Log of DAGMan debugging messages : mydag.dag.dagman.out
   Log of HTCondor library output : mydag.dag.lib.out
   Log of HTCondor library error messages : mydag.dag.lib.err
   Log of the life of condor_dagman itself : mydag.dag.dagman.log

   Submitting job(s).
   1 job(s) submitted to cluster 87274940.
A submitted DAG creates and DAGMan job in the queue

- DAGMan runs on the submit server, as a job in the queue
- At first:

```bash
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?...
OWNER BATCH_NAME SUBMITTED DONE RUN IDLE TOTAL JOB_IDS
alice my.dag+128 4/30 18:08 _ _ _ 0.0
1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?...
ID OWNER SUBMITTED RUN_TIME ST PRI SIZE CMD
128.0 alice  4/30 18:08 0+00:00:06 R 0 0.3 condor_dagman
1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

HTCondor Manual: DAGMan > DAG Submission
Jobs are automatically submitted by the DAGMan job

- Seconds later, node A is submitted:

```bash
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?...

OWNER BATCH_NAME SUBMITTED DONE RUN IDLE TOTAL JOB_IDS
alice my.dag+128 4/30 18:08 _ _ _ 1 5 129.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?...

ID OWNER SUBMITTED RUN_TIME ST PRI SIZE CMD
128.0 alice 4/30 18:08 0+00:00:36 R 0 0.3 condor_dagman
129.0 alice 4/30 18:08 0+00:00:00 I 0 0.3 A_split.sh
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
```

HTCondor Manual: DAGMan > DAG Submission
Jobs are automatically submitted by the DAGMan job

- After A completes, B1-3 are submitted
Jobs are automatically submitted by the DAGMan job

- After **B1-3** complete, node **C** is submitted

```
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?

OWNER  BATCH_NAME  SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice  my.dag+128  4/30 8:08  4  _  1  5  129.0...133.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>?

ID  OWNER   SUBMITTED  RUN_TIME  ST  PRI  SIZE     CMD
128.0 alice  4/30 18:08  0:00:46:36  R  0  0.3  condor_dagman
133.0 alice  4/30 18:54  0:00:00:00  I  0  0.3  C_combine.sh
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
```
Status files are Created at the time of DAG submission

(dag_dir)/

<table>
<thead>
<tr>
<th>A.sub</th>
<th>B1.sub</th>
<th>B2.sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3.sub</td>
<td>C.sub</td>
<td>(other job files)</td>
</tr>
<tr>
<td>my.dag</td>
<td>my.dag.condor.sub</td>
<td>my.dag.dagman.log</td>
</tr>
<tr>
<td>my.dag.dagman.out</td>
<td>my.dag.lib.err</td>
<td>my.dag.lib.out</td>
</tr>
<tr>
<td>my.dag.nodes.log</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*condor.sub* and *dagman.log* describe the queued DAGMan job process, as for any other jobs

*dagman.out* has DAGMan-specific logging (look to first for errors)

*lib.err/out* contain std err/out for the DAGMan job process

*nodes.log* is a combined log of all jobs within the DAG
Removing a DAG from the queue

- Remove the DAGMan job in order to stop and remove the entire DAG:
  ```bash
  condor_rm dagman_jobID
  ```
- Creates a **rescue file** so that only incomplete or unsuccessful NODES are repeated upon resubmission

```bash
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
OWNER   BATCH_NAME   SUBMITTED    DONE  RUN  IDLE  TOTAL  JOB_IDS
alice   my.dag+128  4/30 8:08    4    _  1   6  129.0...133.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended

$ condor_rm 128
All jobs in cluster 128 have been marked for removal
```
Removal of a DAG results in a *rescue file*

(dag_dir)/

<table>
<thead>
<tr>
<th>A.sub</th>
<th>B1.sub</th>
<th>B2.sub</th>
<th>B3.sub</th>
<th>C.sub</th>
<th>(other job files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>my.dag</td>
<td>my.dag.condor.sub</td>
<td>my.dag.dagman.log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my.dag.dagman.out</td>
<td>my.dag.lib.err</td>
<td>my.dag.lib.out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my.dag.metrics</td>
<td>my.dag.nodes.log</td>
<td><strong>my.dag.rescue001</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Named *dag_file.rescue001*
  - increments if more rescue DAG files are created
- Records which NODES have completed successfully
  - does not contain the actual DAG structure
Rescue Files For Resuming a Failed DAG

• A rescue file is created when:
  – a node fails, and after DAGMan advances through any other possible nodes
  – the DAG is removed from the queue (or aborted; covered later)
  – the DAG is halted and not unhalted (covered later)

• Resubmission uses the rescue file (if it exists) when the original DAG file is resubmitted
  – override: condor_submit_dag dag_file -f
Node Failures Result in DAG Failure

- If a node JOB fails (non-zero exit code)
  - DAGMan continues to run other JOB nodes until it can no longer make progress

- Example at right:
  - B2 fails
  - Other B* jobs continue
  - DAG fails and exits after B* and before node C

OSG Summer School 2017
DAGMan > The Rescue DAG
Resolving held node jobs

- Look at the hold reason (in the job log, or with `condor_q -hold`)
- Fix the issue and release the jobs (`condor_release`) - OR - remove the entire DAG, resolve, then resubmit the DAG (remember the automatic rescue DAG file!)

```
$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618>...

ID OWNER SUBMITTED RUN_TIME ST PRI SIZE CMD
128.0 alice 4/30 18:08 0+00:20:36 R 0 0.3 condor_dagman
130.0 alice 4/30 18:18 0+00:00:00 H 0 0.3 B_run.sh
131.0 alice 4/30 18:18 0+00:00:00 H 0 0.3 B_run.sh
132.0 alice 4/30 18:18 0+00:00:00 H 0 0.3 B_run.sh
4 jobs; 0 completed, 0 removed, 0 idle, 1 running, 3 held, 0 suspended
```
DAG Completion

(dag_dir)/

<table>
<thead>
<tr>
<th>A.sub</th>
<th>B1.sub</th>
<th>B2.sub</th>
<th>B3.sub</th>
<th>C.sub</th>
<th>(other job files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>my.dag</td>
<td>my.dag.condor.sub</td>
<td>my.dag.dagman.log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my.dag.dagman.out</td>
<td>my.dag.lib.err</td>
<td>my.dag.lib.out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my.dag.nodes.log</td>
<td>my.dag.dagman.metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*.dagman.metrics is a summary of events and outcomes
*.dagman.log will note the completion of the DAGMan job
*.dagman.out has detailed logging (look to first for errors)
YOUR TURN!
Exercises!

• Ask questions!
• Lots of instructors around

• Coming up:
  – Now-3:15 Hands-on Exercises
  – 3:15 – 3:30 Break
  – 3:30 – 5:00 Workflows 2 & Hands-on