

Intermediate Condor: Workflows

Rob Quick <rquick@iu.edu>

Open Science Grid

Indiana University

Before we begin...

- Any questions on the lectures or exercises up to this point?

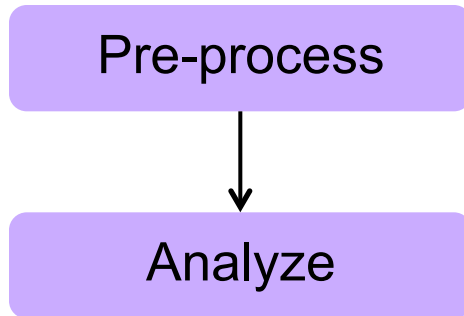


Workflows

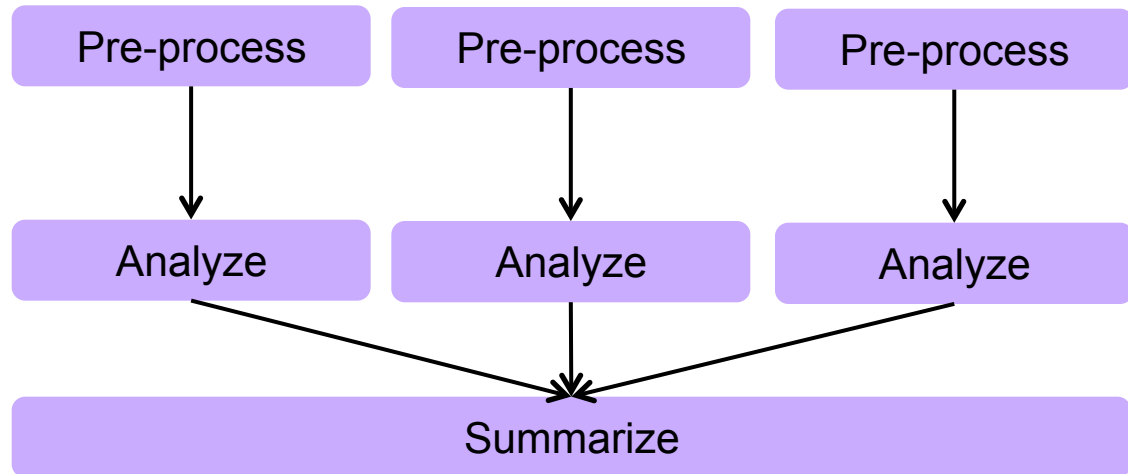
- What if you have a complex set of programs to run for your science?
- For example:
 - You want to analyze a set of images
 - Each image needs to be pre-processed
 - Each image needs to be analyzed
 - You need to summarize the results of all the analyses
 - Each of these is a separate program

Workflows

One Image:



Three Images:



Workflows: definition

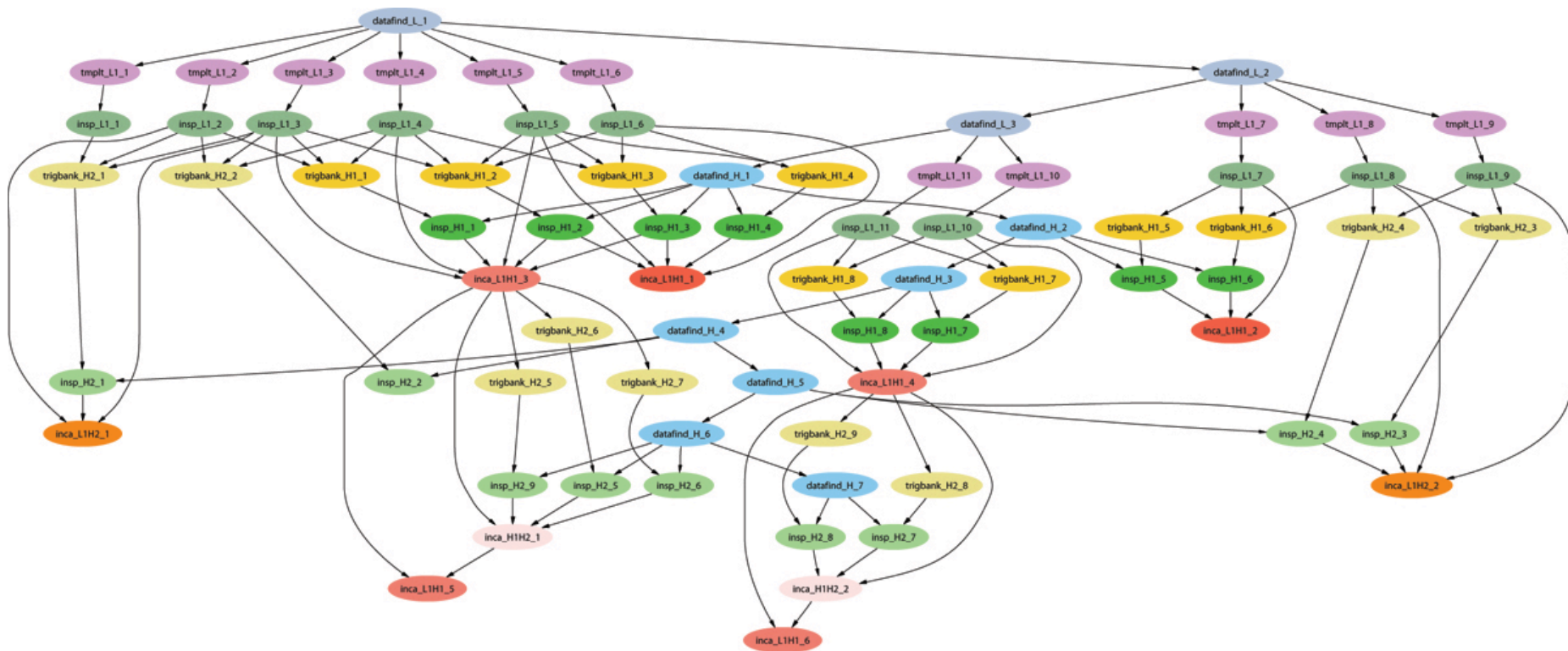
Definition 1:

A set of steps to complete a complex task

Definition 2:

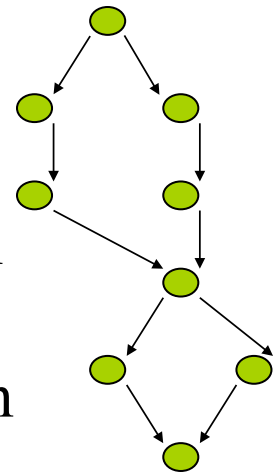
A graph of jobs to run: some jobs need to run before others while other jobs can run in parallel

Example of a LIGO Inspiral DAG



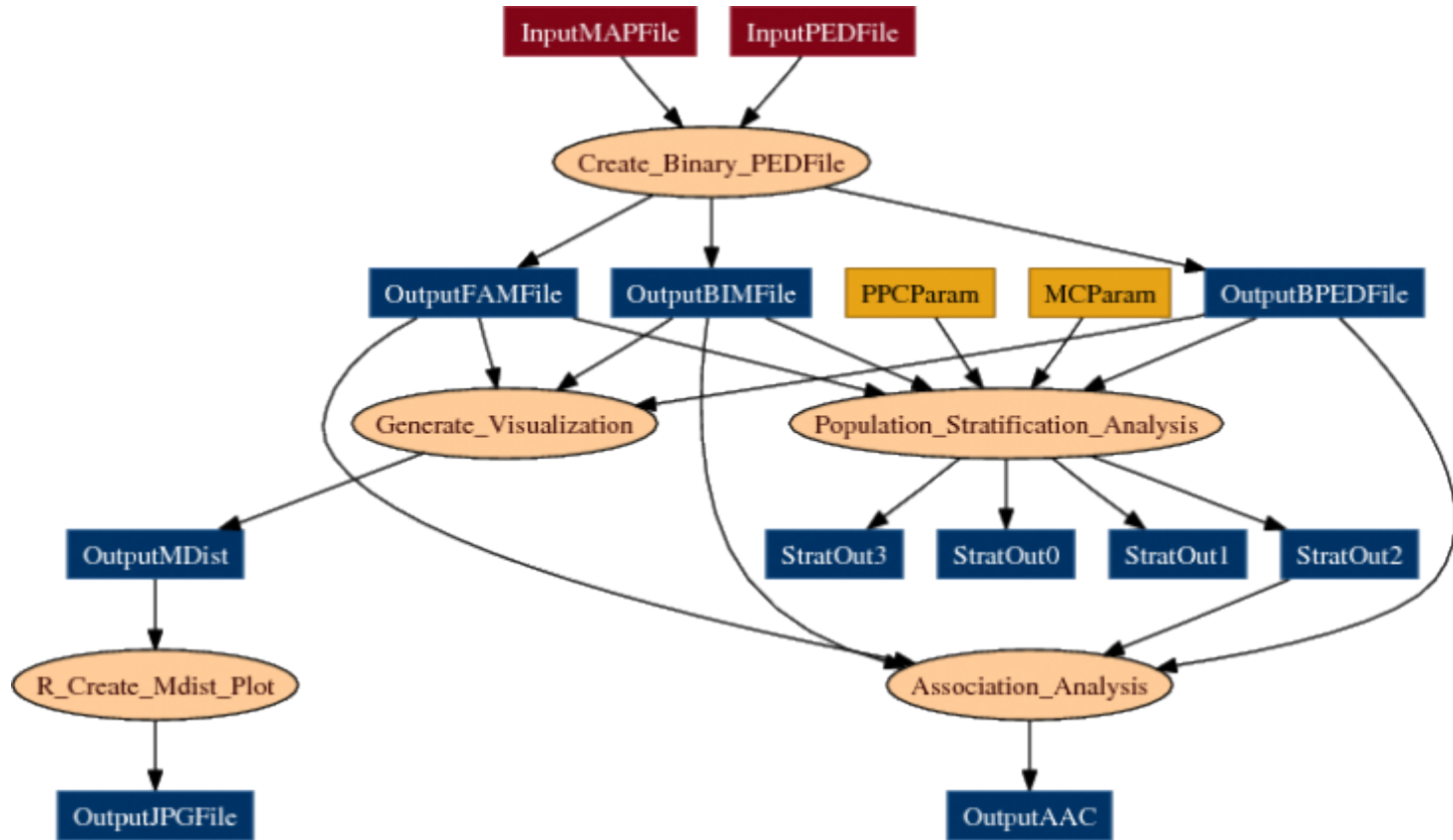
Use of Condor by the LIGO Scientific Collaboration

- Condor handles 10' s of millions of jobs per year running on the LDG, and up to 500k jobs per DAG.
- Condor standard universe check pointing widely used, saving us from having to manage this.
- At Caltech, 30 million jobs processed using 22.8 million CPU hrs. on 1324 CPUs in last 30 months.
- For example, to search 1 yr. of data for GWs from the inspiral of binary neutron star and black hole systems takes ~2 million jobs, and months to run on several thousand ~2.6 GHz nodes.



(Statement from 2010—"last 30 months" isn't from now. Also, I think they do up to 1 million jobs per DAG now.)

Example workflow: Bioinformatics



From Mason, Sanders, State (Yale)

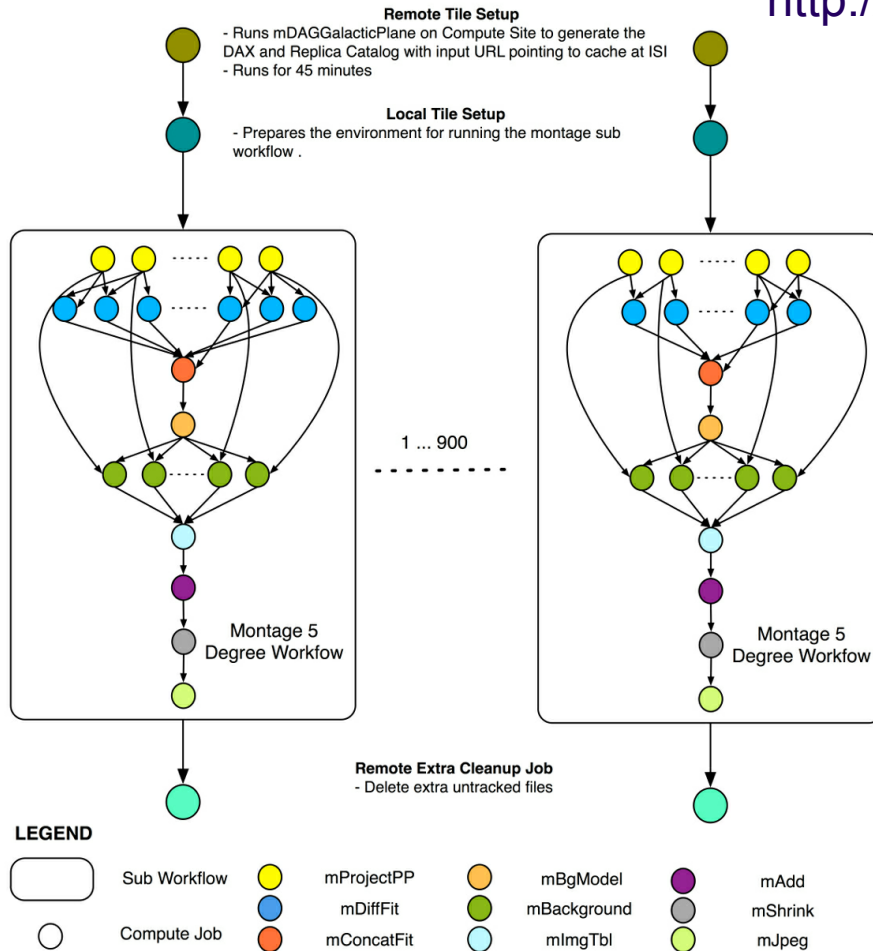
http://pegasus.isi.edu/applications/association_test

Example workflow: Astronomy

Montage Galactic Plane Workflow

From Berriman & Good (JPAC)

<http://pegasus.isi.edu/applications/galactic-plane>



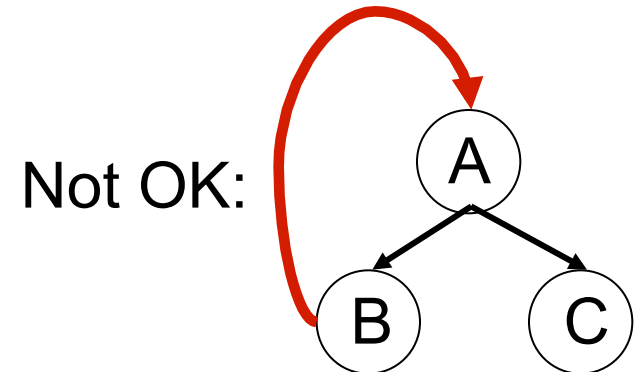
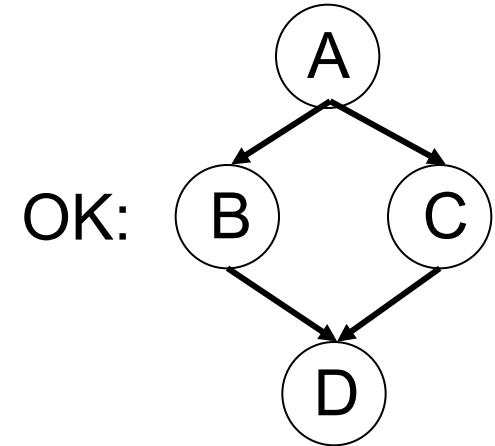
DAGMan

- DAGMan:
Directed Acyclic Graph (DAG)
Manager (Man)
- Allows you to specify the dependencies between your Condor jobs
- Manages the jobs and their dependencies
- That is, it manages a workflow of Condor jobs



What is a DAG?

- A DAG is the 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!

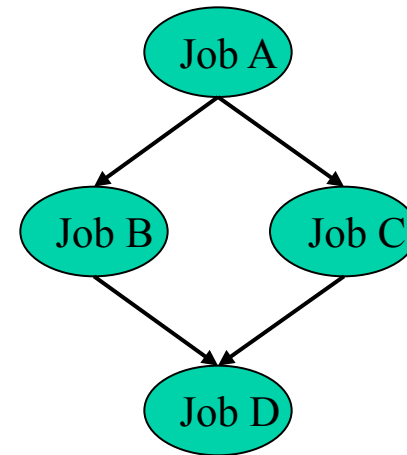


Defining a DAG

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

```
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
```



DAG Files....

- This complete DAG has five files

One DAG File:

```
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
```

Four Submit Files:

```
Universe = Vanilla
Executable = analysis...
```

```
Universe = ...
```

Submitting a DAG

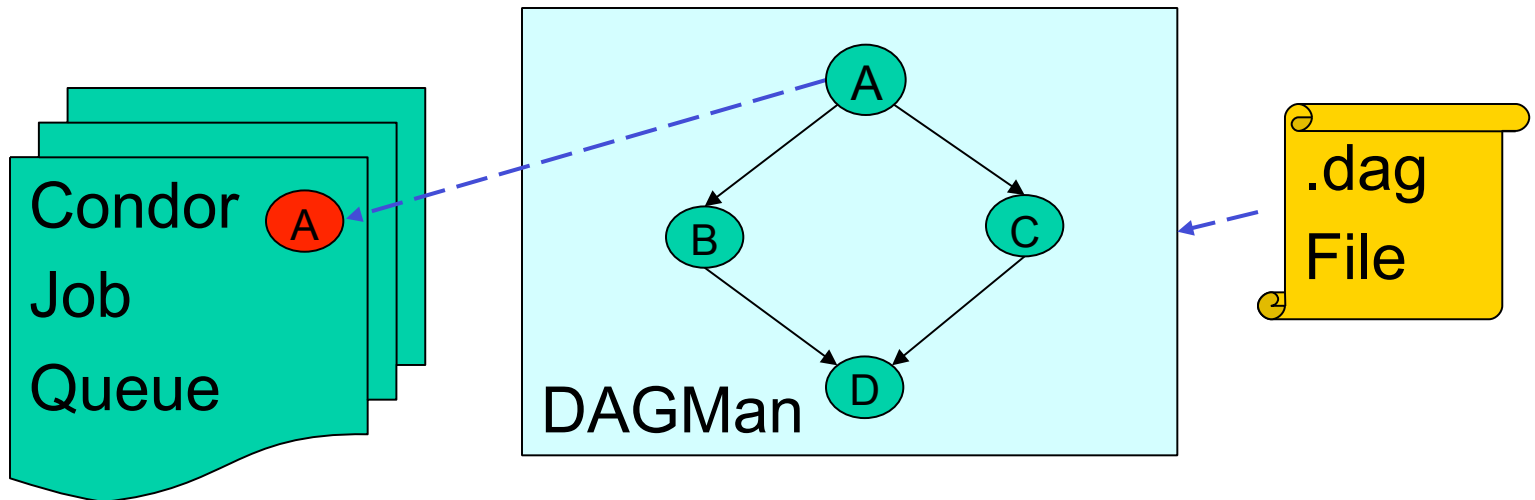
- To start your DAG, just run `condor_submit_dag` with your `.dag` file, and Condor will start a DAGMan process to manage your jobs:

```
% 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

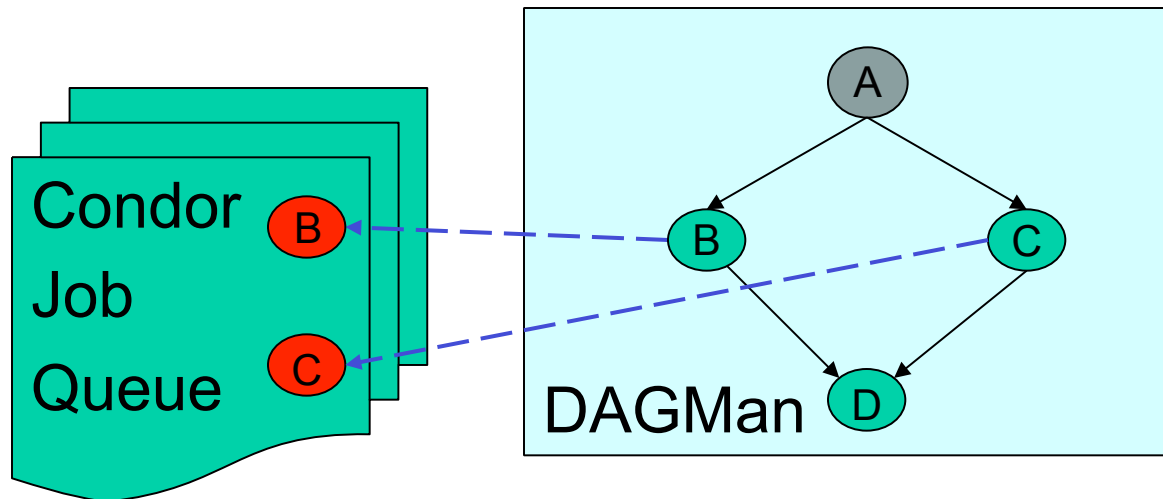
Running a DAG

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



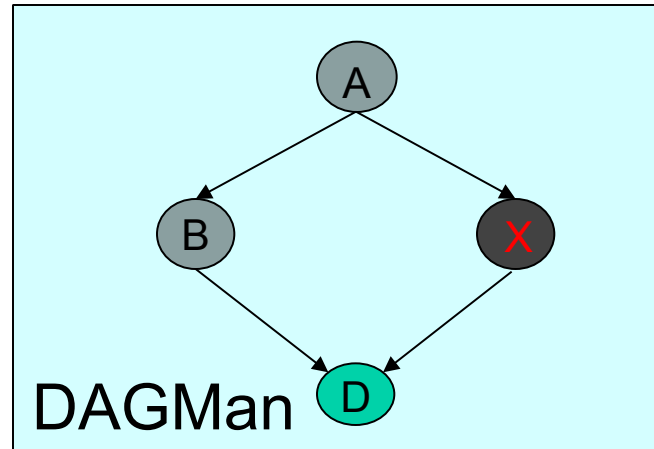
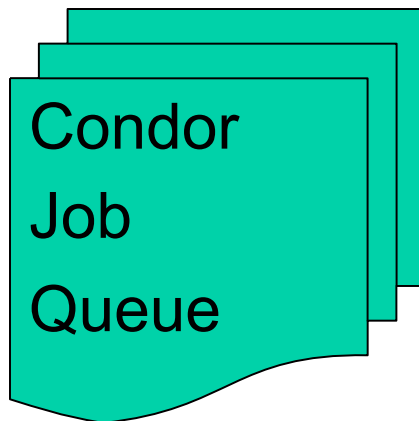
Running a DAG (cont'd)

- DAGMan submits jobs to Condor at the appropriate times
- For example, after A finishes, it submits B & C



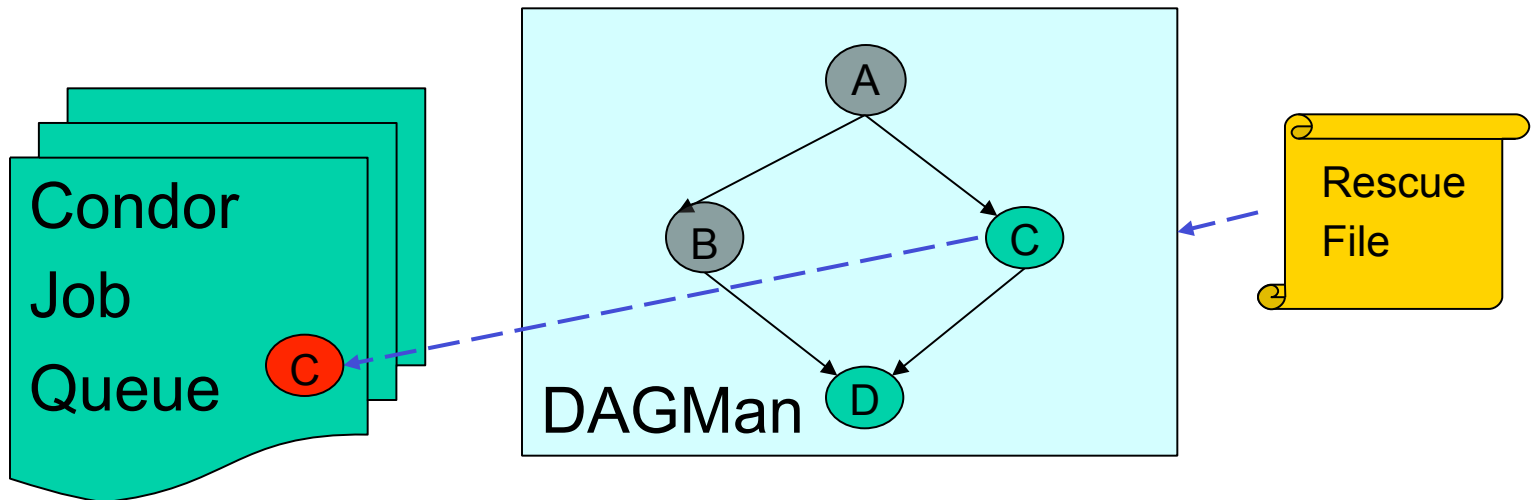
Running a DAG (cont'd)

- A job *fails* if it exits with a non-zero exit code
- In case of a job failure, DAGMan runs other jobs until it can no longer make progress, and then creates a “*rescue*” file with the current state of the DAG



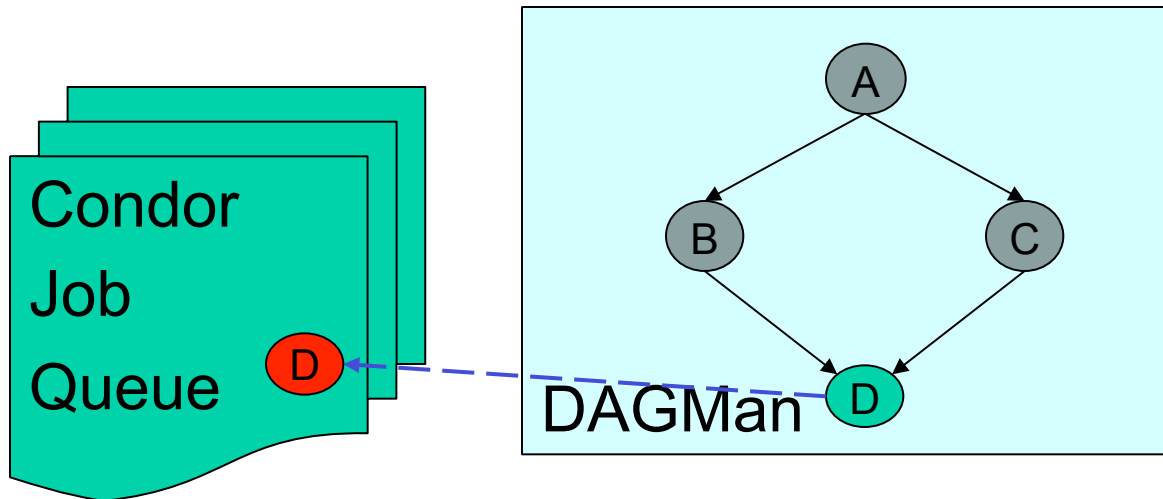
Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG
 - Another example of reliability for HTC!



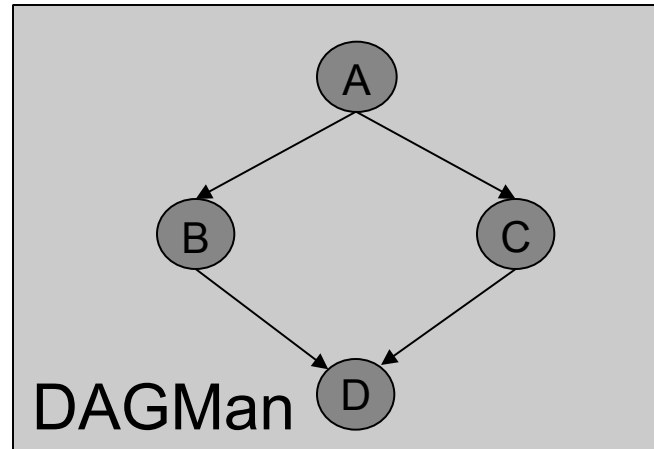
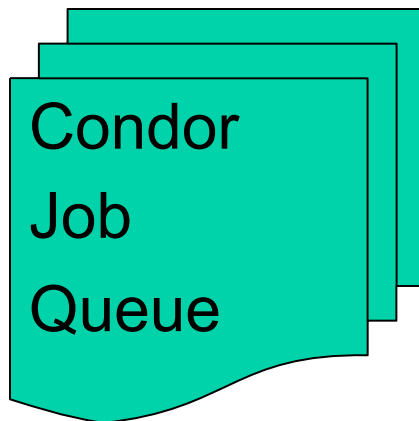
Recovering a DAG (cont'd)

- 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



DAGMan & Fancy Features

- DAGMan doesn't have a lot of “fancy features”
 - No loops
 - Not much assistance in writing very large DAGs (script it yourself)
- Focus is on solid core
 - Add the features people need in order to run large DAGs well
 - People build systems on top of DAGMan

Related Software

Pegasus: <http://pegasus.isi.edu/>

- Writes DAGs based on abstract description
- Runs DAG on appropriate resource (Condor, OSG, EC2...)
- Locates data, coordinates execution
- Uses DAGMan, works with large workflows

Makeflow: <http://nd.edu/~ccl/software/makeflow/>

- User writes *make* file, not DAG
- Works with Condor, SGE, Work Queue...
- Handles data transfers to remote systems
- Does not use DAGMan



DAGMan: Reliability

- For each job, Condor generates a log file
- DAGMan reads this log to see what has happened
- If DAGMan dies (crash, power failure, etc...)
 - Condor will restart DAGMan
 - DAGMan re-reads log file
 - DAGMan knows everything it needs to know
 - Principle: DAGMan can recover state from files and without relying on a service (Condor queue, database...)
- Recall: HTC requires reliability!

Advanced DAGMan Tricks

- Throttles
- DAGs without dependencies
- Sub-DAGs
- Pre and Post scripts: editing your DAG

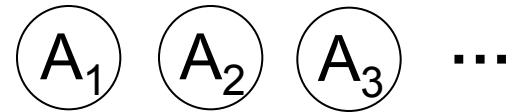
Throttles

- Failed nodes can be automatically retried a configurable number of times
 - Helps recover from jobs that crash some percentage of the time
- Throttles to control job submissions
 - Max jobs submitted
 - Max scripts running
 - These are important when working with large DAGs

DAGs without dependencies

- Submit DAG with:

- 200,000 nodes
- No dependencies



- Use DAGMan to throttle the job submissions:

- Condor is scalable, but it will have problems if you submit 200,000 jobs simultaneously
- DAGMan can help you with scalability even if you don't have dependencies

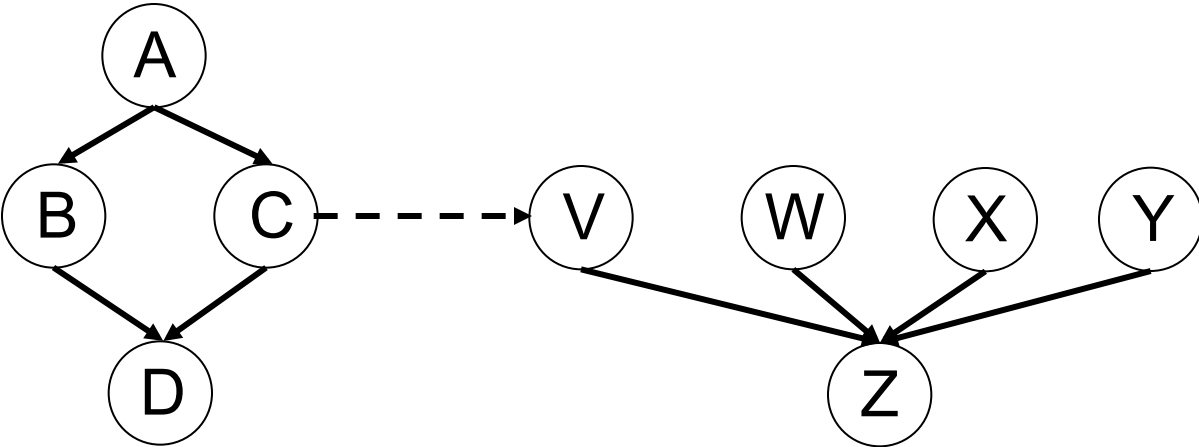


Sub-DAG

- Idea: any given DAG node can be another DAG
 - SUBDAG External Name DAG-file
- DAG node will not complete until sub-dag finishes
- Interesting idea: A previous node could *generate* this DAG node
- Why?
 - Simpler DAG structure
 - Implement a fixed-length loop
 - Modify behavior on the fly



Sub-DAG



DAGMan scripts

- DAGMan allows pre & post scripts
 - Run before (pre) or after (post) job
 - Run on the same computer you submitted from
 - Don't have to be scripts: any executable
- Syntax:

```
JOB A a.sub
```

```
SCRIPT PRE A before-script $JOB
```

```
SCRIPT POST A after-script $JOB $RETURN
```

So What?

- Pre script can make decisions
 - Where should my job run? (Particularly useful to make job run in same place as last job.)
 - What should my job do?
 - Generate Sub-DAG
- Post script can change return value
 - DAGMan decides job failed in non-zero return value
 - Post-script can look at {error code, output files, etc} and return zero or non-zero based on deeper knowledge.

Let's try it out!

- Exercises with DAGMan.





Questions?

- Questions? Comments?
- Feel free to ask me questions later:
Rob Quick - rquick@iu.edu