

Intermediate HTCondor: More Workflows

Monday pm

Greg Thain

Center For High Throughput Computing
University of Wisconsin-Madison

Before we begin...

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



Advanced DAGMan Tricks

- Throttles
- DAGMan Variables
- Retries
- DAGs without dependencies
- Sub-DAGs
- Pre and Post scripts: editing your DAG
- SPLICes: DAGs as subroutines

Throttles

- Throttles to control job submissions
 - Max jobs idle
 - `condor_submit_dag -maxidle XX work.dag`
 - Max scripts running
 - `condor_submit_dag -maxpre XX -maxpost XX`
- Useful for “big bag of tasks”
 - Schedd holds everything in memory



DAGMan variables

```
# 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
```

DAGMan variables (Cont)

```
# Diamond dag
Job A a.sub
Job B a.sub
Job C a.sub
Job D a.sub

VARs A OUTPUT="A.out"
VARs B OUTPUT="B.out"
VARs C OUTPUT="C.out"
VARs D OUTPUT="D.out"

Parent A Child B C
Parent B C Child D
```

DAGMan variables (cont)

```
# a.sub  
Universe = vanilla  
Executable = gronk  
  
output = $(OUTPUT)  
  
queue
```



Retries

- Failed nodes can be automatically retried a configurable number of times
 - Helps when jobs randomly crash

```
Job A a.sub
```

```
Job B b.sub
```

```
Job C c.sub
```

```
Job D d.sub
```

```
RETRY D 100
```

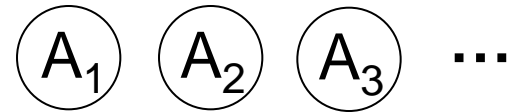
```
Parent A Child B C
```

```
Parent B C Child D
```


DAGs without dependencies

- Submit DAG with:

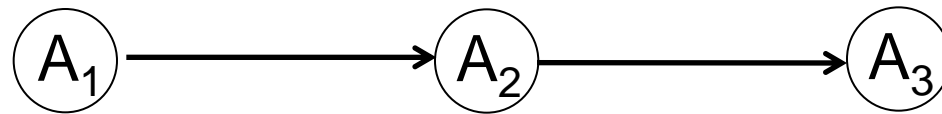
- 200,000 nodes
- No dependencies



- Use DAGMan to throttle the job submissions:

- HTCondor 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

Shishkabob DAG



- Used for breaking long jobs into short
- Easier for scheduling

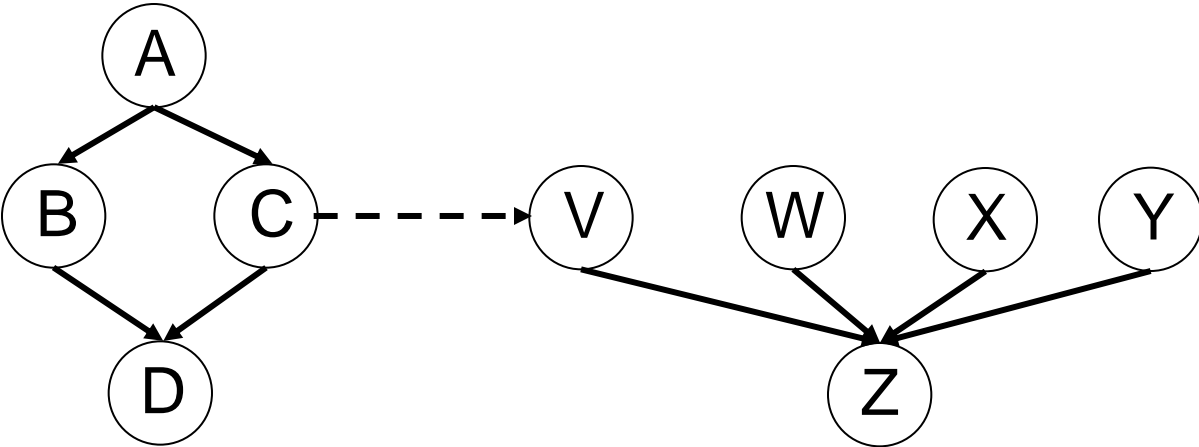


Sub-DAG

- 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
```

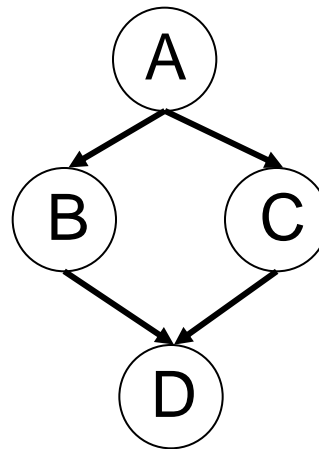
```
SCRIPT PRE A before-script $JOB
```

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

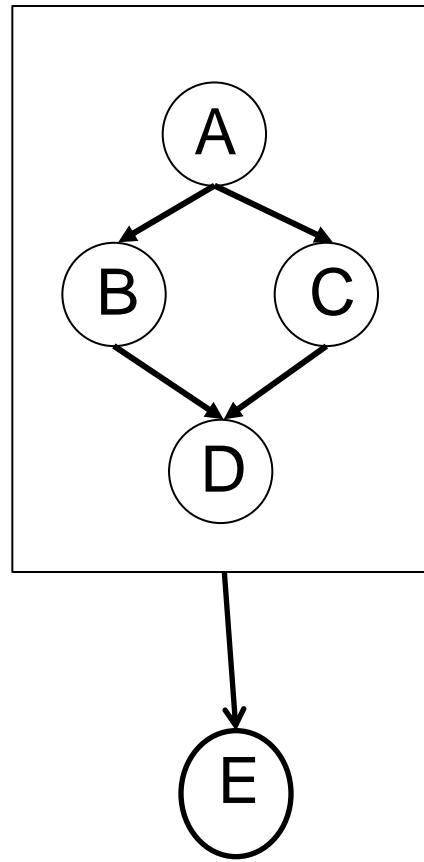
So What?

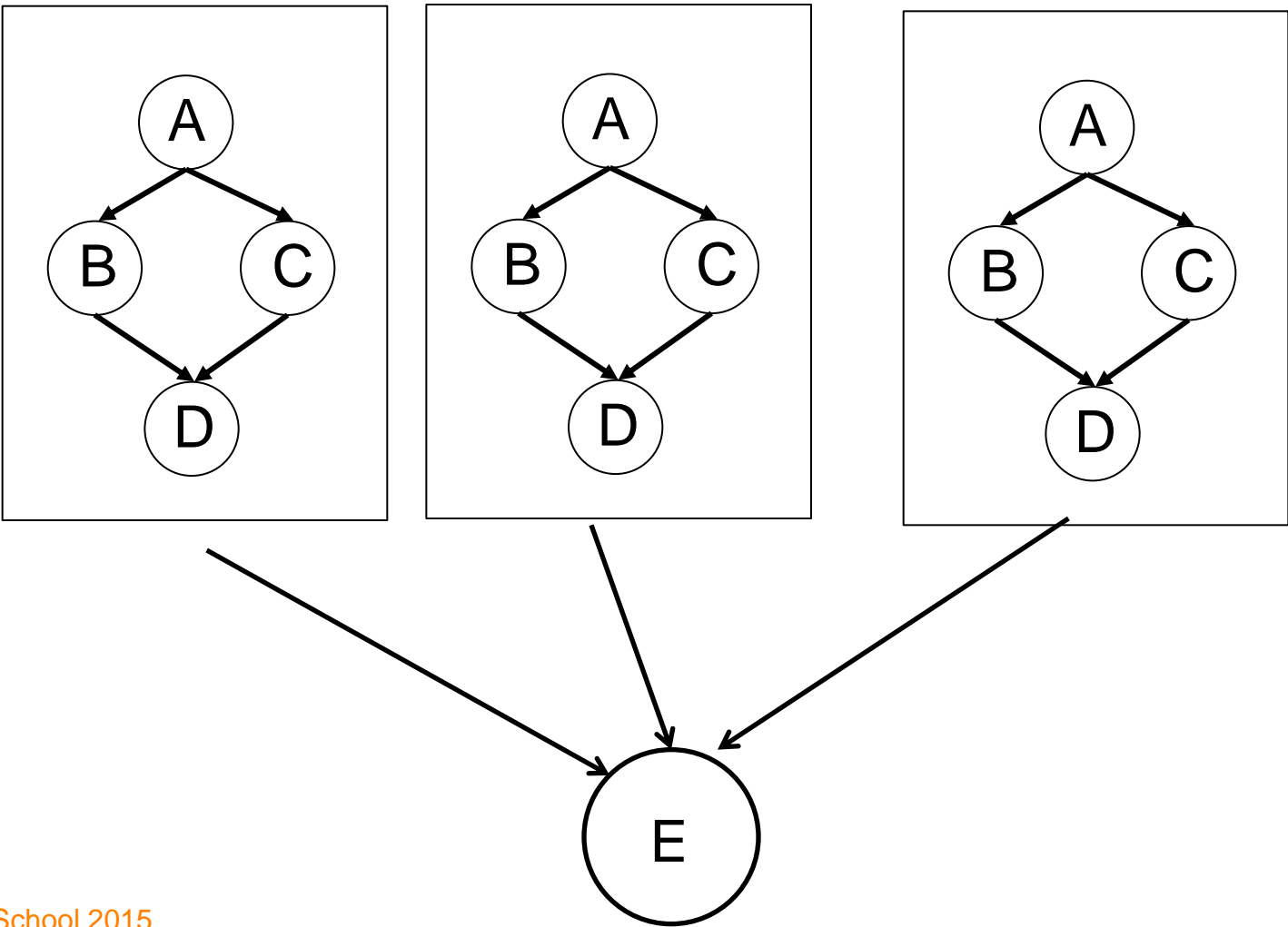
- Pre script can make decisions
 - Where should my job run? (Particularly useful to make job run in same place as last job.)
 - What should my job do?
 - Generate Sub-DAG
- Post script can change return value
 - DAGMan decides job failed in non-zero return value
 - Post-script can look at {error code,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

SPLICEs: DAGs as subroutines



SPLICEs: DAGs as subroutines





SPLICE Syntax

```
SPLICE name dagfile.dag
```

Creates new node with dag as node

CHILD / PARENT / etc all work on ndoes



Example

JOB E E.submit

SPLICE DIAMOND diamond.dag

PARENT DIAMOND CHILD E

Let's try it out!

- Exercises with DAGMan.



Questions?

- Questions? Comments?
- Feel free to ask me questions later: