Incorporating dependencies in the schedd

# Overview

Workflows, or pipelines, are crucial to efficiently using distributed resources in the high throughput setting. Such pipelines must offer the ability to specify granularity of jobs in order to schedule jobs efficiently and also must guarantee, as much as possible, the reliability of the computation. In HTCondor, we have the workflow solution of DAGman, and other pipeline solutions, such as Makeflow, have been developed for working with workflows by other groups

DAGMan submits jobs, which HTCondor then executes. HTCondor writes events into the UserLog while a job is submitted. For the purposes of this document, a job can be four states: Idle, Running, Completed, or Held. This document is most interested in when a job is instantiated in the Idle state and when it transitions from Running to Completed, which is taken as the terminal state. A job dependency is between one or more parents and one or more children. The workflow rule implemented by DAGMan states that a job is submitted in Idle state when all the dependencies of a child job have been satisfied; that is, when all the parent jobs have transitioned to Completed state. This means that DAGMan is the only entity within HTCondor which has all the dependency information about the workflow. That is, the schedd has only a snapshot view of the jobs in the workflow: the schedd only has state information for jobs which are currently in Idle, Running, and Held states, but it has no state information about jobs that have Completed or that are yet to be submitted.

The proposal here is to extend, in a limited way, the amount of information the schedd has about job dependencies. In particular, we want to encourage users to use alternative meta-schedulers such as DAGMan for “large” or “complex” DAGs, while eliminating the use of them for “trivial” DAGs. The boundaries of “large”, “complex”, and “trivial” will be taken to be ambiguous here, as they depend on the sophistication of the user and the resources available to the schedd itself.

## Architecture of Proposed Solution

This development will require changes in the schedd and submit.

In the schedd, we will want to keep track of parent-child relationships. For each job, the schedd will maintain a list of child nodes (nodes that are dependent on this node), and a list of parent nodes (nodes that this node depend on). The schedd will write into the jobad the list of parent nodes, but not the list of child nodes; tools should be able to build the list of dependencies from a query of the job ads, to answer the question, “Why has this job not begun running?’’

For condor\_submit, the notion of parent jobs will be added. Condor\_submit will verify that the jobids of parent jobs have a smaller job id than the child jobs (that is, all child jobs will have a cluster id that is at least that of their parents, and, if the clusterId’s are the same, then the process id will be greater than that of its parent). The schedd will also have to check this, to prevent cyclic dependencies. This is to provide a guarantee that no cycles will be submitted, wherein a parent is a child of itself or of its descendants.

We are not discussing DAGMan implementation; DAGMan is orthogonal to this development. DAGMan will not use the functionality we are discussing here. This is a way for users to specify trivial job dependencies easily.

The submit directive parents will be introduced, as will new submit file macros PREVCLUSTER and PREVPROCESS. A new classad attribute, DependsOn, will also be introduced. For the traditional diamond dag, we will have a submit file as follows. The following is a skeletal example to highlight the new syntax being introduced.

executable = a.sh

queue

A = $(PREVCLUSTER).$(PROCESS)

executable = b.sh

parents = $(A)

queue

B = $(PREVCLUSTER).$(PREVPROCESS)

executable = c.sh

parents = $(A)

queue

C = $(PREVCLUSTER).$(PREVPROCESS)

executable = d.sh

parents = $(B),$(C)

queue

condor\_submit will interpret this submit file as follows: first it will submit job a. Job a will enter the queue normally: except as otherwise determined, it will be in idle state; the schedd will negotiate for it, and it will run. Jobs b and c will then be submitted, with no guarantee on the order of submission. They will also be submitted and the schedd will not negotiate for them until a completes. It will probably be best to indicate somehow that the jobs are “waiting for parents to finish”. Jobs b and c will have a DependsOn attribute in their classAd which will be a string attribute containing a comma-separated list of job ids that are the parents of b and c (in this case, a). If a user does a condor\_qedit command affecting the DependsOn attribute, then the schedd will need to modify its tables to reflect the change. The schedd will ignore changes to DependsOn if the job has been matched to a claim. A log message will need to be output to indicate that an attempt to change the DependsOn attribute has been made and been ignored. If a completes and exits the queue before b and c are submitted, b and c will be idle. In other words, if a parent job exits the queue before a child job is submitted, the parent is assumed to have successfully completed. Job d will then be submitted, again with an indication that it is “waiting for parents to finish”. When a completes, b and c will be released, and so on. If a leaves the queue with a non-zero exit code or is condor\_rm’d, then b, c, and d will be removed. Similarly, if b fails, then d will be removed. Should c then be allowed to complete? I don’t know. Since DAGMan would allow c to run to completion, perhaps it should be allowed to run

Another development will be the following submit file

Executable = a.sh

Queue 1000

A = $(PREVCLUSTER)

Executable = b.sh

Parents = $(A).$(PROCESS)

Queue 1000

The idea is that each job I the “b” cluster will have a dependency on the corresponding process in the “a” cluster. It looks as though this will fit into the scheme above with no changes.

One unresolved question is what to do about spooled input. It seems like the simplest course is to not allow jobs to depend on jobs that are spooled. In fact, I am wondering if having parents should necessarily imply the skip\_filechecks is set to true, since otherwise submit will complain about missing files (if the files are built in a previous part of the pipeline, before the executable runs and after submit completes).

# Development Plan

The above changes to submit should take 3 days to write and test. Changes to submit to handle the hold codes for job dependencies should also take about 3 days to write and test. It seems from the above that condor\_q will also need to examine the DependsOn attribute to check whether a job is waiting for parents to complete; this should take 2 days.