HTCondor Tutorial

Presented by: Jon Wedell BioMagResBank wedell@bmrb.wisc.edu

Background

During this tutorial we will walk through submitting several jobs to the HTCondor workload management system. We will then investigate how the jobs run in the HTCondor environment and explore how to manage submitted jobs.

Your first job

Within your summer workshop folder, in the `htcondor` directory you should see a collection of files, several of which end in `.sub`. These are HTCondor submission description files. These files describe a job to be submitted to HTCondor and contain the information that HTCondor needs to run the job. We will go over the parameters in these files in detail, but first let's submit a job to HTCondor that we will come back to later. To submit a job, use the `condor_submit` command as such:

```
condor_submit crondor.sub
```

Crondor

The job we just submitted is an example of a "crondor" job. HTCondor has the ability to schedule jobs to run at a specific time, and it also has the ability to have a given job run an infinite number of times at the scheduled interval. Therefore we can use it in place of the traditional Linux command scheduling program `cron`. The benefit of using HTCondor is that on the chance that the previous invocation of the job is still running HTCondor is smart enough to delay the next scheduled invocation until the previous job has completed. This is useful if having the same job running twice simultaneously could cause issues.

If you haven't already, you will shortly see a message pop up on your screen. That message was the application we specified to run in the crondor.sub file. The relevant bits of that file for this exercise are:

```
cron_minute = */3
cron_hour = *
cron_day_of_month = *
cron_month = *
cron_day_of_week = *

on_exit_remove = False
```

The `cron_minute` directive specifies that the job should be ran every 3 minutes. The other time related constraints are left satisfied under any conditions, but could optionally be added to, for example, only have our job run every 3 minutes on Mondays.

The `on_exit_remove` argument will ensure that once the job completes it will run again rather than be removed from the queue.

The Queue

At this point we will take a brief detour from looking at the HTCondor submit file to learn how to manage the HTCondor queue. First, to see what execution slots are available in your pool, run the `condor_status` command. You should see a result similar to the following:

```
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot10@wiscdaily.n LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:05
slot1@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:34:37
slot2@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:05
slot3@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:06
slot4@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:07
slot5@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:08
slot6@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:09
slot7@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:10
slot8@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:03
slot9@wiscdaily.nm LINUX      X86_64 Unclaimed Idle      0.000 1598  0+01:35:04
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        X86_64/LINUX    10     0       0        10       0          0        0

               Total    10     0       0        10       0          0        0
```

This indicates that there are 10 slots in the pool, and they are all currently available to run our jobs.

To see what jobs are actually in the queue, use the `condor_q` command. You will see an output similar to the following:

```
-- Schedd: wiscdaily.nmrbox.org : <127.0.0.1:51028?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   1.0   jwedell         7/10 14:59   0+00:00:00 I  0   0.1  zenity --info --te

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

This shows the job that we submitted to the pool earlier in the tutorial. At this point the recurring pop-up messages that are generated by the job you submitted are probably becoming irritating. Let's remove that job so that the messages stop.

To remove a job from the queue, use the `condor_rm` command followed by the job ID. You can also run the command followed by your Linux user ID to remove *all* of your jobs. Use the latter option with caution.

Based on the information returned by the `condor_q` command as shown above, the following command will remove our periodic crondor job:

```
condor_rm 1
```

The submit file

While we examined some optional arguments to the submit file earlier, lets look at a file in detail to see what arguments are necessary for a submission.

Here is the most minimal example of a submit file:

```
universe = vanilla
executable = /bin/ls
arguments = /

log = logs/basic.log
output = logs/basic.out
error = logs/basic.err
queue
```

Going through the lines in order:

- The universe specified which HTCondor universe to run in. These are discussed in the slides. The most straightforward is the `vanilla` universe. As a beginner this universe should be suitable for all your jobs.
- The executable specifies which command should actually be ran when the job runs. You must use an absolute path.
- The arguments line, while technically optional, will almost always be used. It allows you to specify which command line arguments to provide to the executable.
- The log, output, and error commands specify where HTCondor will log. The output and error will be populated with whatever the job writes to STDERR and STDOUT while the log will contain the HTCondor logs related to the scheduling and running of the job.
- The queue parameter is special. Each time this argument is encountered in the file HTCondor will submit a job to the queue with whatever arguments were specified prior in the file. You can specify queue multiple times on different lines, and optionally change any of the parameters between the queue lines to customize the various runs of the job. You can also provide

a number after `queue` to submit multiple jobs with the parameters. (e.g. if doing a Monte Carlo simulation)

You can see an example of this behavior used in practice in the “shasum.sub” file.

Requirements

At this point, you should submit the job specified in the requirements.sub file with the following command:

```
condor_submit requirements.sub
```

Looking at that submit file, the new lines you will notice are the following:

```
request_cpus = 1
request_disk = 1MB
request_memory = 10GB
```

These arguments speak for themselves. They are telling the HTCondor matchmaker that in order to run your job the machine must be able to provide the specified resources. If you are running this job on the 2017 summer workshop and you looked at the `mem` line when running `condor_status` earlier you may have noticed that none of the slots in the pool had more than 8GB of memory available. This is indeed a problem as this job requires 10GB.

If you look at the condor queue (`condor_q`) you will see that the job you just submitted is not running. To examine why HTCondor is not running the job, you can use the `-analyze` argument to `condor_q`. For a job ID of x, the command to run is:

```
condor_q -analyze x
```

You will get an output from HTCondor describing the number of slots in the pool, and why your job is not running on any of them. If it is able to, HTCondor will also suggest which requirement you need to change in order for your job to run.

In the case of this job, you will see the following suggestions:

```
    Condition                         Machines Matched    Suggestion
    ---------                         ----------------    ----------
1   ( TARGET.Memory >= 10240 )        0                   MODIFY TO 1598
2   ( TARGET.Arch == "X86_64" )       10
3   ( TARGET.OpSys == "LINUX" )       10
4   ( TARGET.Disk >= 1024 )           10
5   ( ( TARGET.HasFileTransfer ) || ( TARGET.FileSystemDomain == "wiscdaily.nmrbox.org" ) )
                                      10
```

By looking at the "Suggestion" column you can see that only the memory requirement needs to be changed.

You should be wary of requesting more resources than your job needs in order to ensure that it has as many slots as possible to run on.

Output files and variables

Finally, submit the `multiple.sub` file the same way you submitted your other jobs to HTCondor.

First to note is that this submit file is using a variable in some of the arguments. The variable it uses, `$(PROCID)` is defined automatically by condor. It is the process ID. Each time you submit to HTCondor, the job is assigned a job ID. Furthermore, while each time queue is specified in the file creates an additional job, they share they same job ID, it is the process ID that is incremented. When looking at the queue, you can see these numbers in the form "jobID.procID". Running `condor_rm` with a job ID will remove all of the jobs with that ID, running `condor_rm` with a jobID.procID will only remove the specific instance of the job with the given procID.

What this means is that for this submit file the `touch` command will run five times, each with a different `$(PROCID)`, and therefore the arguments for the jobs that HTCondor fills in will be different wherever we use the `$(PROCID)` variable. By putting it in the "arguments" section we are providing the process ID to each instance of the `touch` command, which will create files with the name of the process ID.

Files

In the vanilla universe, HTCondor will automatically transfer back any files created during the execution of a job to your local machine. (Though it will not transfer back folders created or files within them automatically - you must manually specify those using a "transfer_output_files argument if you want them to be preserved.)

At this point the jobs should have completed, and we can look in the output directory to see what is present. `ls output/` should return `0 1 2 3 4`. This indicates that the `touch` command ran five times with five different process IDs and created one file for each invocation

Logs

Submit the `shasum.sub` file, wait 60 seconds, and then look in the logs directory at the files `logs/shasum.out` and `logs/shasum.err`. In the error file, you will see the aggregated output of the STDERR stream for all processes in this job. In the output file, you will see the aggregated STDOUT stream.

Summary

This has been a very brief demonstration of the basic features of HTCondor job submission and management. For more details on the submit file format, please see the HTCondor documentation at: http://research.cs.wisc.edu/htcondor/manual/v8.7/2_5Submitting_Job.html