CONDOR: howto

di Miguel Villaplana
v1, settembre 2017

First steps

HTCondor [1] is open-source software created and released by the Center for High Throughput Computing at UW-Madison in the US. You are also encouraged to use the official documentation [2] which will always be the reference. This guide exists to give an overview, and also to show some details to help with the system in the University of Milan.

To start using HTCondor you first need to login to a machine that is configured to submit jobs to the HTCondor batch system. A list of the HTCondor pools currently available in University of Milan and their corresponding HTCondor submit nodes is shown here[3] under the tab **HPC Clusters**.

The command condor_status is a versatile tool that may be used to monitor and query the HTCondor pool.

```
$ condor_status -available
```

Shows available slots

```
$ condor_status -available -autoformat Name Memory Cpus Disk
```

Shows name, memory, cpus, disk space of available slots

```
$ condor_status -run
```

Shows slots which are currently running jobs

```
$ condor_status -constraint 'RemoteUser == "<username>@<your_pool_name>"'
```

Displays slots running your jobs.

Submit file

HTCondor jobs cannot be submitted directly on the command line. Each job (or set of jobs) requires a submit file. Here's an example submit file:

```
executable              = hello_world.sh
arguments               = $(ClusterID) $(ProcId)
queue
```

Let's go through that line by line.

executable: The script or command you want HTCondor to run.

arguments: Any arguments that could be passed to the command. We're using interpolated values here. ClusterId will normally be unique to each submit file. ProcId is incremented by one for each job in each cluster. In this simple example, which is defining a single job, the value of ProcId will be 0.

queue: This schedules the job. It becomes more important (along with the interpolation) when queue is used to schedule multiple jobs by taking an integer as a value.

Universe

HTCondor allows submission to different platforms or architectures with the use of something it calls universes. In the submit file, you can specify the universe, like so:

```
universe = vanilla
```

The vanilla universe is the default that you will get if you specify nothing, and is in general the one that you'll want to use. However, we will run other universes, such as the parallel universe.

Input, output and logs

Most jobs require input and output, and HTCondor will also log the execution of a job or jobs. The directives are fairly self-explanatory:

```
input  = jobinput.txt
output = joboutput.txt
log    = joblog.txt
```

Whereas HTCondor doesn't require a shared file system, the use of one enables some additional features, such as being able to use condor_wait on the log file to monitor job state transitions. In the above example, the paths are relative, but are presumed to be on a shared file system that both the submission node and the condor scheduler can both access.

There are a number of variables that can be used in the generation of the filenames, which is useful when a submit file is being used to generate multiple jobs. This is an example of such a submit file excerpt:

```
input  = input/job.$(ClusterId).$(JobId).txt
output = output/job.$(ClusterId).$(JobId).output
log    = log/job.$(ClusterId).$(JobId).log
```

The variables that can be used in the submit file are detailed here which is useful to separate files. It is though also sometimes useful to have a single log file for each submission, in which case condor_wait can watch individual jobs in the log file.

Requirements

In HTCondor the job's ClassAd can describe requirements, for example the name of the execute machine, the operating system is specified, etc. The requirements should be written correctly, so be aware that attribute names are insensitive but for the string values of the ClassAd, comparison operators like <, >, <=, =>, == compare case insensitively or use the special operators =**?**= and =**!**= to compare case sensitively. Specifying requirements can be used to target particular parts of the batch system. This should be used with care, by definition it means limiting the nodes that are available for your job. However it is sometimes necessary, for instance to override the default OS platform that we set for jobs and to run on CentOS7:

```
requirements = (OpSysAndVer =?= "CentOS7")
```

Requirements can use any ClassAd, and expressions can be chained. The logical operators can be used are: **&&** for AND, and || for OR.

```
requirements = ( (OpSysAndVer =?= "CentOS7") && (Arch =="INTEL") )
```

Submitting the job

On a properly configured submit host you simply need to run the condor_submit command:

```
$ condor_submit hello.sub
Submitting job(s).
1 job(s) submitted to cluster 70.
```

You will note the reference again to "cluster", this output shows the "ClusterId" referred to in the submit file. Normally, you get one cluster per run of condor_submit.

Submitting multiple jobs

The queue directive can take an integer to submit multiple jobs, for example with the following submit file:

```
executable            = runmore.sh
input                 = input/mydata.$(ProcId)
arguments             = $(ClusterID) $(ProcId)
output                = output/hello.$(ClusterId).$(ProcId).out
error                 = error/hello.$(ClusterId).$(ProcId).err
log                   = log/hello.$(ClusterId).log
queue 150
```

The "queue 150" directive instructs condor to run 150 jobs. Note that each job will have an incremental $ProcId, which can be used to interpolate other directives in the submit file, such as to separate input files.

Sometimes it is easier to group a set of jobs and execute them as an array than execute them one by one. HTCondor provides the ability to match executable files using regular expressions and add a job to the queue for each file.

```
executable              = $(filename)
arguments               = $(ClusterId)$(ProcId)
output                  = output/$(ClusterId).$(ProcId).out
error                   = error/$(ClusterId).$(ProcId).err
log                     = log/$(ClusterId).log

queue filename matching files *.sh
```

Note: In the submit description file, $(filename) is interpreted to be the file matching the regular expression.

Monitoring the job

The command condor_q can be used to see the current status of the jobs in the queue:

```
$ condor_q
-- Schedd: bigbird01.cern.ch : <128.142.194.108:9618?...
    OWNER   BATCH_NAME       SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
    bejones CMD: hello.sh   10/3  14:08      _      _     1      _ 70.0

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

The condor_q command provides information regarding the current state of the jobs, the name of the schedd, the name of the owner, etc.

The progress of a job can be followed by executing:

```
$ watch condor_q
```

A constraint can be used to make condor_q displaying only your jobs.

```
$ condor_q  -constraint 'OWNER == "<username>"'
```

The -nobatch option can be used to show the status of each individual job rather the cluster summary.

```
$ condor_q -nobatch

-- Schedd: bigbird04.cern.ch : <128.142.194.115:9618?... @ 03/28/17 17:13:42
ID        OWNER           SUBMITTED     RUN_TIME ST PRI  SIZE CMD
21847.0   fprotops        3/28 17:13   0+00:00:00 I  0    0.0 welcome.sh
```

More information regarding the life cycle of the job can be found in the log file. It contains information about the submitted jobs in chronological order such as which machine executed the job, if the job terminated correctly, if the job aborted, etc.

```
001 (2465.000.000) 12/07 15:18:17 Job executing on host: <188.185.177.87:9618?
  addrs=188.185.177.87-9618+[--1]-9618&noUDP&sock=2869_c9b5_3>
...
006 (2465.000.000) 12/07 15:18:18 Image size of job updated: 1
0  -  MemoryUsage of job (MB)
0  -  ResidentSetSize of job (KB)
...
005 (2465.000.000) 12/07 15:18:18 Job terminated.
  (1) Normal termination (return value 0)
        Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
28  -  Run Bytes Sent By Job
47  -  Run Bytes Received By Job
28  -  Total Bytes Sent By Job
47  -  Total Bytes Received By Job
Partitionable Resources :    Usage  Request Allocated
   Cpus                 :                1         1
   Disk (KB)            :       15        1    501507
   Memory (MB)          :        0     2000      2000
```

The command condor_wait displays the progress of the job by watching in the submission's log file.

```
$ condor_wait -status log/hello.70.log
70.0.0 submitted
 70.0.0 executing on host <188.185.180.233:9618?addrs=188.185.180.233-9618+
 [--1]-9618&noUDP&sock=23729_b2e3_13>
 70.0.0 completed
 All jobs done.
```

The option -better-analyze performs a detailed matchmaking analysis to determine how many slots are available to run the requested job.

```
$ condor_q  -better-analyze  <job_id>
```

Note: this command is specially useful to spot jobs failing due to wrong requirements in the submit file.

The command condor_tail displays the last lines of STDOUT of a running job

```
$ condor_tail <job_id>
```

Managing a job

A job can be removed from the queue at any time by using the condor_rm command.

```
$ condor_rm <job_id>
```

A job can be put on hold state with the condor_hold command. When a job is put on hold, it will not be scheduled to run until it is released. If the job is running when condor_hold is invoked, it will be vacated from the machine it was running on.

```
$ condor_hold <job_id>
```

A job can be released with the condor_release command. When a job is released from the hold state, it is returned to idle state, and will be scheduled to run when possible. Only jobs that are on hold can be released.

```
$ condor_release <job_id>
```

The priority of owned jobs can be changed with the condor_prio command. The priority of a job can be any integer, with higher numbers corresponding to greater priority. For adjustment of the current priority, + <value> increases the priority by the amount given with <value>, while - <value> decreases the priority by the amount given with <value>.

```
$ condor_prio  + <value> <job_id>
```

To create an ssh session to a running job, type

```
$ condor_ssh_to_job name <schedd> <job_id>
```

Notice that you must be the owner of the job. The remote session runs with the same user as the running job, in the folder from which it has been launched. A PID, associated to the job, is provided. To leave the session, type **logout**. More options can be found through condor_ssh_to_job help.