Submitting Batch Jobs with HTCondor

Useful links:

- Condor manual
- Condor manual: submit a job
- Video tutorial

Command overview

Command	What it does
condor_submit	Submit new Jobs
condor_status	View Pool Status
condor_q	View Job Queue
condor_q -analyze	Why job/machines fail to match
condor_q_edit	Edit job attributes
condor_submit -i	Submit interactive job
condor_hold/release	Hold a job, or release a held job
condor_run	Submit and block
condor_rm	Remove Jobs
condor_prio	Intra-User Job Prios
condor_history	Completed Job Info

condor_q, condor_rm, condor_history, condor_q_edit and *condor_release* have similar selections

- <user>
- *ClusterID.ProcID/ClusterID*
- *-constraint 'classadd expression'* e.g. *-constraint 'RemoteJob=?=True'*

Job submission

You have to submit your jobs from portal machines: **portal.darwin.kit.edu**

To submit jobs with HTCondor, you have to describe them using a JDL file (Job Description Language). This file contains both instructions for the job itself, e.g. which executable and arguments to use, and meta-information, e.g. how much RAM you need, which universe and on which cloud site it should run.

Job environment

All jobs running in HTCondor run inside a container. This enables to run your jobs on each resource in the same software environment. Currently, we provide SLC6, SLC7, CS8 and AlmaLinux9 environments. The standard environment is the SLC7 environment. If your job needs another environment provided by another docker image, you can also define it via ClassAd. Example for SLC7 environment:

```
Universe = docker
docker_image = mschnepf/slc7-condocker
```

in grid-control via

```
JDLData = ... docker_image=mschnepf/slc7-condocker
```

If you use some special libraries, we recommend compiling against libraries provided by cvmfs (e.g. /cvmfs/sft.cern.ch/…). The docker container has most of the installed libraries on the portal machines, but not all.

Recommended docker images:

OS	Image name
SLC7	mschnepf/slc7-condocker
:::	cverstege/cc7-gridjob (updated version, can be used when having problems with remote storage access with the mschnepf/slc7-condocker image)
RHEL8	mschnepf/cs8-base
ALMA9	cverstege/alma9-gridjob

Job priority

In general, machines are allocated based upon a user‘s priority. A lower numerical value for user priority means higher priority, so a user with priority 5 will get more resources than a user with priority 50. User priorities in HTCondor can be examined with the ’‘condor_userprio’’ command.

Users can give their own jobs a **job priority** which controls which jobs should be processed earlier.

More detailed information can be found in the HTCondor Manual.

Detailed job requirements and attributes

In order to run your jobs on appropriate resources, it is advisable that you define some job requirements and/or attributes. Here is a description on how this is done in JDL files. If you are using grid-control, some of these values are taken directly from the configuration file, others have to be entered identically in the `[condor]` section. HTCondor distinguishes between attributes and requirements:

Requirements

Requirements restrict the applicable machine pool by introducing restrictions (“only run my job on machines which …”).

Restricting this pool can also positively influence match-making/priority: Machines that are well equipped to handle IO-sensitive jobs will prefer them over jobs that only require CPU.

Requirements are identical for Grid-Control and HTCondor.

Requirements	Comment
ProvidesIO	for IO-intensive jobs
ProvidesCPU	for CPU-intensive jobs

Attributes

Attributes have additional information about the job, for example, if it is a remote-capable job (i.e. you handle input/output via remote storage or HTCondor’s built-in file-transfer). In contrast to requirements, attributes supply *optional* information that **can** be used by the batch system for match-making. Usually, they allow you to access additional resources and/or match you better to the appropriate resources where your job will run in the most appropriate environment.

Grid-control uses the configuration‘s values translated to HTCondor ClassAds for the “Request…” class. This means you do not have to explicitly define them in the ’‘[condor]’’ configuration part.

We also provide multicore jobs. For that set the ClassAd `RequestCPUs` to the requested amount of CPU-cores for your job. We optimized our HTCondor system for a single (default) and eight CPU-core jobs.

Attribute	Comment
+RequestWalltime	maximum runtime of one job (in seconds)
RequestMemory	maximal needed memory of one job (in MB)

ClassAds

HTCondor works with information in form of ClassAds. ClassAds are key value/expression pairs, e.g.

- *Owner = "username"*
- *START = TARGET.Owner=?="username"*
- *requirements = (TARGET.MEMORY>=RequestMemory)*

Every instance in HTCondor has ClassAds: jobs, worker nodes, scheduler,…

- get ClassAds of jobs via *condor_q/condor_history*
- get ClassAds of daemons via *condor_status*
- -/ shows all ClassAds of the instance
- *-af ClassAd* shows only the ClassAd value/expression

Important ClassAds:

- *requirements*: requirements of the job to slot
- *CloudSite*: group of worker nodes with similar setup
- *HoldReason*: reason why your job is on hold
- *RemoteUserCPU*: amount of CPU time the job used on a worker node

ClassAd syntax:

- keys are not case sensitive, only strings in quotes are case sensitive
- values can be *undefined*
- is equal (*=?=*) returns only false or true while *==* can return false, true, undefined
- is not equal (*=!=*)
- and (*&&*), or (*||*)

Example

Executable

Create a simple shell script, for example run.sh:

```
# ! / bin / bash
##### WN informaton
echo -e " Hostname : $ ( hostname ) "
echo -e " user : $ ( whoami ) "
echo -e " spawndir : $ (pwd) "
##### setup environment
source /cvmfs/belle.cern.ch/tools/b2setup release-09-00-01
##### start workload at spawndir and create output.root with results
basf2 /work/$(whoami)/HTCondor_Tutorial/code/01 _simple_submit/mc_example.py -o output.root
##### copy output from WN
cp output.root/work/$(whoami)/HTCondor_Tutorial/code/01_simple_submit/simple_submit_output.root
```

Make it executable:

<code bash> chmod +x run.sh </code>

The executable should be tested on portal machine before submission. Things to have in mind:

- check for errors
- check for resource via htop: CPU, number of processes, Memory, …
- develop portable code
- job runtime should be longer than 20 min
- output file size should be bigger than 1 GB

Submit file

Create a submit file submit_file.jdl:

```
universe = container
```

```
##container environment
container_image = /cvmfs/unpacked.cern.ch/registry.hub.docker.com/cverstege/alma9-gridjob:latest

##executable
executable = ./run.sh

#Resource demand
request_cpus = 1  ## CPU cores (default 1)
request_memory = 2000  ## RAM (default 2000 MB)
request_disk = 500000  ## Disk (default 500 000 kB)
+RequestWalltime = 86400  ## Walltime (default 86 400 sec)

##logging
log = log.log
stdout = stdout.log
stderr = stderr.log

##submit queue
queue
```

It is advised to store the logs in your `/work` directory.

Submit the Job

Submit the job with:

<code bash> condor_submit submit_file.jdl </code>

You should see a message like:

<code> Submitting job(s). 1 job(s) submitted to cluster 12345. </code>

Job Status

Check the status of your jobs: ‘’condor_q ’’

To see detailed information for a specific job: `condor_q 12345 -long`

FAQ

How can I rerun held jobs?

Check first the hold reason of your held job with `condor_q jobID -af HoldReason` and change some ClassAds with `condor_qedit jobsID ClassAd newValue`. A hold job can be released with `condor_release (jobID / user)`

If the hold reason is “`Job exceeded maximum number of execution attempts.`” you have to set some variables for that hold jobs. To release these jobs, run this script: release_jobs.sh

```
#!/bin/bash

condor_qedit ${1} -constraint 'JobStatus=?=5' JobRunCount 1
condor_qedit ${1} -constraint 'JobStatus=?=5' NumJobMatches 1
condor_qedit ${1} -constraint 'JobStatus=?=5' NumJobStarts 1
condor_qedit ${1} -constraint 'JobStatus=?=5' NumShadowStarts 1
condor_qedit ${1} -constraint 'JobStatus=?=5' NumShadowExceptions 1
condor_release ${1}
```

How can I start multi-core jobs?

Our HTCondor instance has a different kind of worker nodes. All worker nodes support minimal 4 core jobs. To run a job with more cores set in the JDL file `request_cpus = <number of cores>`. Please make sure that your program does not run on more than the requested cores!

Why don’t my jobs start running?

Condor decides when and where to run your jobs based on job attributes, such as desired wall time or memory, and worker node attributes, such as available memory or policies. To find out why your job is not starting, run `condor_q -analyze` . If any jobs are broken, run `condor_q -held` to read their error message.

How do I submit several jobs?

If you want to submit several jobs with the same executable but different arguments, e.g., input file or seed, the `queue` command enables to submit a cluster of jobs. For example, `submit.jdl`:

```
Universe = container
container_image = /cvmfs/unpacked.cern.ch/registry.hub.docker.com/cverstege/alma9-gridjob:latest

executable = . /executable.sh
arguments = $(ProcID)

request_cpus = 1
request_memory = 2000
request_disk = 500000

log = log_$(ProcID).log
stdout = stdout_$(ProcID).log
stderr = stderr_$(ProcID).log

queue 5
```

in the example above each job has the same *ClusterID*, but a different *ProcID*. Job ID is *ClusterID.ProcID*. *ClusterID* and *ProcID* can be used in submit file.

You can also read variables from a file (e.g. input.csv), on line per job:

```
Universe = container
container_image = /cvmfs/unpacked.cern.ch/registry.hub.docker.com/cverstege/alma9-gridjob:latest

executable = . /executable.sh

request_cpus = 1
request_memory = 2000
request_disk = 500000

log = log_$(ProcID).log
stdout = stdout_$(ProcID).log
stderr = stderr_$(ProcID).log

queue arguments from input.csv
```

How do I transfer data via HTCondor?

There are different ways of transferring files:

- Transfers via HTCondor runs through the scheduler. This can be inefficient and extra load on portal machines.
- Transfer files to the WN: *transfer_input_files*
- Transfer files from the WN at the end of the job: *transfer_output_files = output.root*
- Rename output files: *transfer_output_remaps ="output.root = output/output_$(Process).root"*

Do not use *transfer_output_files* or *transfer_output_remaps* for files bigger than 10MB.

What is considered a data intensive job?

Jobs with more than 2 GB input files per 30 min runtime are data intensive. Write on local storage on the WN and copy the results at the end of the jobs. For some Grid storage requires a VOMS (Virtual Organisation Membership Service) proxy