HTCondor Essentials

31.10.2017

Index

Login

To login use ssh protocol through the following command:

```
> ssh -i <path_private_key> <username>@frontend.recas.ba.infn.it
```

The login occurs in the user home:

```
> pwd
/lustrehome/<username>
```

Top

How to submit a job in the HTCondor pool

Suppose we want to run the echo command to print on screen the string “Hello World!”.
To submit a job in HTCondor we need a submit description file, a text file which will describe the job (path of executable, requirements, etc...).
A further explanation about submit description file and its commands is described later.

Submit description file

```
# This is a comment in the submit file
# file name : submit_echo
# universe describes an execution environment. Set universe to vanilla
universe = vanilla
# Path of the executable (it can be a system command, your own application, a
script, etc...)
executable = /bin/echo
# The argument to pass to the executable
arguments = "Hello World!"
# The output of the remote machine running the job will be printed on
echo.out
output = echo.out
error = echo.error
log = echo.log
request_cpus = 1
rank = Memory
queue
```

`Note:` if you don't specify `request_cpus`, `request_disk`, `request_memory` in the submit file, the job will run on a default amount of cpus (1), memory and disk space.

To submit the job use `condor_submit` passing as an argument the name of the submit file:

```
> condor_submit submit_echo -name ettore
```

Top

Why the `-name<schedd>` option?

In order to submit HTCondor job-related commands you need to specify a schedd, hence the `-name<schedd>` option in the command submission above.

All commands described in this document need the `-name ettore` option to work except `condor_status`.

Top

Submitting a job

`condor_submit` is the program for submitting jobs for execution under HTCondor.

```
> condor_submit <path_submit_file>
```

```
> condor_submit <path_submit_file> -interactive
```

Indicates that the user wants to run an interactive shell on the remote machine that execute the job.

The submit file should look like the following examples:

Submit description file example in order to execute the mathematica application with 1GB of RAM as resource allocation policy:

```
executable          = mathematica
universe            = vanilla
input               = test.data
output              = loop.out
error               = loop.error
log                 = loop.log
request_memory      = 1 GB
initialdir          = run_1
queue
```

Submit description file example in order to execute the mathematica application with 24GB of RAM and 8 cores as resource allocation policy:

```
executable          = mathematica
universe            = vanilla
input               = test.data
output              = loop.out
error               = loop.error
log                 = loop.log
request_memory      = 24 GB
request_cpus        = 8
initialdir          = run_1
queue
```

Resources allocation are managed in a hard way. By configuration HTCondor will not allow the job to exceed the requested resources.

Requesting more than one core could lead the job to high waiting time in queue.

Top

Checking status of submitted jobs

Getting id and other info about a job

```
> condor_q
```

Displays information about jobs in the HTCondor job queue. For every job is shown:

- id: id of the condor job
- status: I – idle (waiting for a machine to execute on), R – running, H – on hold, S – suspended, C – completed, X – removed, < – transferring input, > – transferring output
- other info (like runtime, size, priority (PRI), name of the executable (CMD))

Top

Displaying running jobs

```
> condor_q -run
```

Displaying only your jobs

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

Displaying jobs on hold

```
> condor_q -hold
```

Displaying a specific job

```
> condor_q <job_id>
```

Top

Why a job won't complete its execution?

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

Performs a detailed matchmaking analysis to determine how many slots are available to run the requested job.

Top

Checking output of a running job

```
> condor_tail <job_id>
```

Displays the last lines of `stdout` of a running job

Top

Managing a job

Removing a job from the queue

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

```
> condor_rm <job_id>
```

Top

Putting a job on hold

A job can be put on hold state with `condor_hold` command.
When a job is put on hold, it will not be scheduled to run until 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>
```

Top

Releasing a job

A job can be released with `condor_release` command.
When a job is released from 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>
```

Top

Changing the priority of the jobs

The priority of jobs can be changed with `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*. - *value* decreases the priority by the amount given with *value*.

```
> condor_prio -p <+|-value> <job_id>
```

Top

Checking pool status

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

`Note:` `condor_status` command doesn't need `-name<schedd>` option to work.

```
> 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

Top

Displaying slots running your jobs

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

Top

SSH to a running job

To create an ssh session to a running job, type

```
> condor_ssh_to_job -name <schedd> <job_id>
```

replacing `<schedd>` with the name of the schedd to which the job has been submitted (e.g. ettore), and `<job_id>` with the 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
```

Further information and some examples on how to use it can be found at this link.

Top

Submit description file

A submit description file is a text file which contains everything HTCondor needs to know about the job to be submitted such as the name of the executable to run, the initial working directory, command-line arguments, etc...

Mandatory commands

`universe = vanilla`
: `universe` defines an HTCondor execution envirorment. Set this command to `vanilla`.

`executable = <path_name>`
: Path where the executable is located.

`queue`
: This command will send the job to the queue, so it should be the last command in the submit description file.

Top

Really useful commands

`input = <path_name>`
: This file should contains the keyboard input the program requires (this file is basically `stdin`). End of line character is equivalent to press ENTER on the keyboard.

`output = <path_name>`
: This file contains the output the program writes on the remote machine (this file is basically `stdout`).

`error = <path_name>`
: This file contains any error messages the program would normally write to the screen (this file is basically `stderr`).

`log = <path_name>`
: This file contains info about what happens while the job is running. If this file is specified HTCondor will add a log entry for several event like when the job starts running, migrates to another machine, the job completes.

`arguments = <arguments list>`
: List of arguments to be supplied to the executable as part of the command line.
In HTCondor there are two possible formats to specify arguments, known as the old syntax and the new syntax. You can find more details and examples here.
If you type the following command:
`> condor_q <job_id>` **`-long -autoformat`** `args arguments`
depending on which sintax you used, it will be prompted on screen `args` if you used the old sintax, `arguments` if you used the new sintax.

`priority = <+|- value>`
: Assign a priority to the job. The highest the value, the greater the priority.

Top

Adding requirements to the machines which will execute the job

`request_cpus = <num_cpus>`
: Requested amount of CPUs to execute the job. If not specified, the number of CPUs for the job is 1.

`request_disk = <kilobytes>`
: The requested amount of disk space in KB requested for this job. If not specified, the job will run on a default amount of disk space.

`request_memory = <megabytes>`
: The requested amount of memory in MB requested to run the job. If non specified, the job will run on a default amount of memory.

`requirements = <ClassAd expression>`
: Run the job on the machines that match the ClassAd expression.

`rank = <ClassAd expression>`
: The `rank` command will assign an order to the machines based on the ClassAd expression and will run the job on the machine with the highest rank. the `rank` command and requirements commands described above can coexist in the same file: while the requirements commands will exclude a machine which doesn't meet the user criteria from running the job, the `rank` command will run the job on the best machine, according to the user criteria, currently available.

Top

About `requirements` and `rank`

Both `requirements` and `rank` need to be valid ClassAd expression. In a ClassAd expression attribute names are case insensitive while string values are always case sensitive. So the following ClassAd expressions are valid:

```
Requirements = OpSys == “LINUX” && Arch == “INTEL”
requirements = opsys == “LINUX” && arch == “INTEL”
```

while the following expression is not:

```
requirements = opsys == “linux” && arch == “intel”
```

Top

More Job Examples

1. Multiple submission

The following submit file executes 5 istances of the program `sleep`; for every run the process id related to the single execution is supplied as argument:

Submit description file

```
universe      = vanilla
executable    = /bin/sleep
arguments     = $(Process)
sleep         = sleep.log
queue 5
```

`Note:` The job id in HTCondor is defined as `Cluster.Process`. If you submit more than one job from the same submit file each job will share the same cluster id.

Top

2. More about multiple submission

The following submit file executes 3 instances of the program `echo` and for every run a different string is printed on `stdout`

Submit description file

```
universe      = vanilla
executable    = /bin/echo
output        = job_$(Cluster).$(Process).out
error         = job_$(Cluster).$(Process).err
log           = job_$(Cluster).log

arguments     = "String 1"
queue

arguments     = "String 2"
queue

arguments     = "String 3"
queue
```

For the x run:

- `stdout` will be sent to `job_idcluster.x.out`
- `stderr` will be sent to `job_idcluster.x.err`

Top

3. Working with files

Because the pool uses a shared file system you don’t have to specify in the submit file the transfer of file to and from the remote machine that execute the job. All you need to do is to make sure that the paths of the files (either absolute or relative) the program needs are valid.
Suppose we want to run a simple bash script that prints on screen the content of a text file:

```
#!/bin/sh
while read line; do
    echo "$line"
done < file.txt
```

Submit description file

```
universe      = vanilla
executable    = script.sh
log           = script.log
output        = script.out
queue
```

Top

4. Multiple submission and files

Suppose we want to run 2 instances of an executable named `myprogram` that needs input from `stdin`, a text file and one argument to work; once its execution is complete, it yields an output file named `out.txt`
Using the command `initialdir` allows to specify a different base directory for each job avoiding in this example the overwriting of the output file `out.txt`

Submit description file

```
universe      = vanilla
executable    = myprogram
log           = myprogram.log
input         = myprogram.in
output        = myprogram.out
error         = myprogram.err
initialdir    = run_1
arguments     = a
queue
initialdir    = run_2
arguments     = w
queue
```

For the x run:

- `stdout(stderr)` will be sent to `run_x/myprogram.out(run_x/myprogram.err)`
- `stdin` will be read from `run_x/myprogram.in`

Top