Migration from LSF to HTCondor

Notes by D. Bozzato

CERN, Geneva

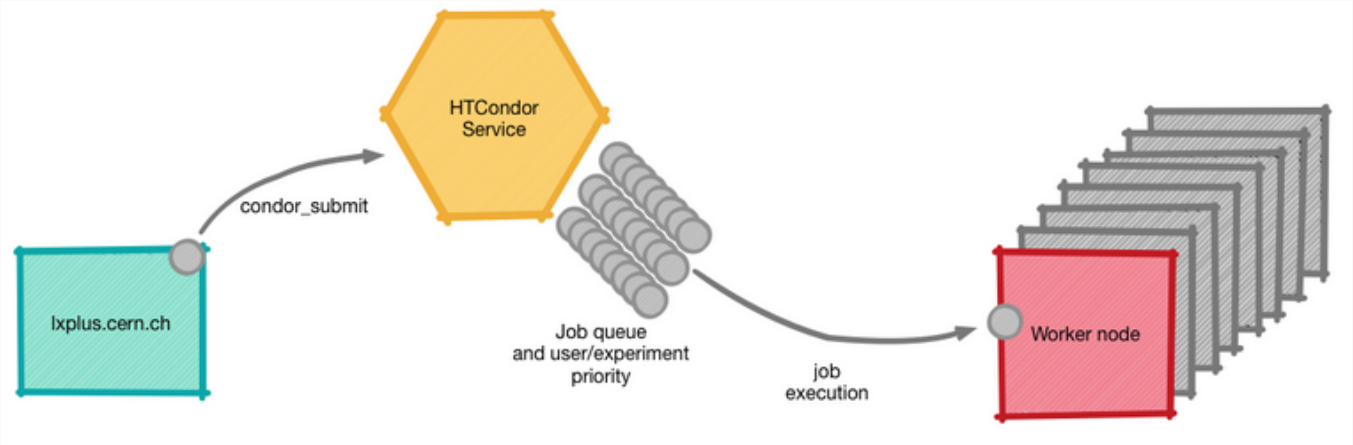
Table of contents

Introduction

Job Pattern

A *job* is a discrete unit of work comprising an executable and some requirements like memory or operating system. The job is accepted by the batch system and is put in a queue. The system is fair-share meaning that the larger the agreed share, the more jobs one gets to run and the more the service is used the more likely is that the next job to run will belong to someone else.

Data flows I



Jobs are typically submitted from the `lxplus.cern.ch` Interactive Linux Service. This has access to AFS and EOS file systems. The executable for a job can be taken both from a path in AFS and a path in EOS. The user submits a job to HTCondor Service and once a job is scheduled for execution, the batch system passes its definition to a spare slot on an execution host (or worker node).

Data flows II

The job starts and normally continues to run until it is complete, at which point the user is notified (by default) and the job log, standard out and standard error are written.

At any time, a user can query the batch service for the status of his or her jobs, seeing which are completed, which are running and which are still waiting in the queue.

User priorities I

The fair-share for each group is based on quota and subsequently, HTCondor calculates the users' priorities that they belong to the same group. According to the number of resources that a user is using, the priority changes. The fair-share balancing is only one of the factors to be take into account. The matching of the job with the worker node is related also to the amount of available resources, job's requirements, etc.

Machines are allocated to users based upon a user's priority. A lower numerical value for user priority means higher priority and the share is inversely related to the ratio between user priorities: a user with priority 10 will receive twice as many machines as a user with priority 20.

User priorities II

The priority of each individual user changes according to the number of resources the individual is using. If the number of machines a user currently has is greater than the user priority, the user priority will worsen. If the number of machines is less then the priority, the priority will improve by numerically decreasing. The speed at which HTCondor adjusts the priorities is controlled with the configuration variable `PRIORITY_HALFLIFE` (one day as default). If a user that has user priority of 100 and is utilizing 100 machines removes all his/her jobs, one day later that user's priority will be 50, and two days later the priority will be 25.

User priorities III

The command `condor_userprio` displays users' priorities.
Information shown contain:

- Real priority: depends on user resource usage.
- Priority factor: its a number that is used to create different classes of users.
- Effective priority: is calculated by multiplying the real priority by the priority factor.
- Res Used: number of resources currently used

Limits

There is no limit for job submissions per user. The maximum number of jobs that can run simultaneously is 10.000.

Jobs will be removed if not finished by specified run-time. They will be automatically removed also if they are in hold state for more than 24 hours or if they restarted more than 10 times.

Getting started with HTCondor

Prerequisites

A machine that is configured to submit jobs to the HTCondor batch system is required: essentially it means logging into lxplus.cern.ch

The two main ingredients to run a job are the submit file (*.sub file) and an executable. In order to submit the job, one has to run the command condor_submit like:

```
condor_submit file_to_submit.sub
```

Jobs can be queried running the command `condor_q`.

The command `condor_rm <user>` removes all the jobs submitted by the user.

Submit file

The submit file is a file containing everything that HTCondor should know in order to run a job. A typical submit file has the form:

```
executable              = exe.sh
arguments               = $(ClusterId)
output                  = /path/to/output/output.out
error                   = /path/to/error/error.err
log                     = /path/to/log/logfile.log

+JobFlavour             = ”espresso”
request_cpus            = 1
initial_dir             = /path/to/initial/dir
transfer_input_files    = file1, file2

queue
```

Submit file: `executable` and `arguments`

The executable is the script or the command that one wants HTCondor to run. It can be a simple shell script (like in the example above), a python script, etc.
Arguments are any arguments that could be passed to the command and that are needed to specified executable. In the example above the cluster number *ClusterId* is passed to the executable `exe.sh`

Submit file: `output`, `error` and `log`

These are where STDOUT, STDERR of the command or script should be written to and output of logs for submitted job should be written to. These can be a relative or absolute path.
WARNING: HTCondor won't create the directory (as LFS used to do) so if the directory specified in the submit file is not present an error message will be given and the job will not be submitted.

Submit file: `Job Flavours` I

In order to help scheduling the jobs, a maximum run time should be set so a job can be assigned a *flavour*. Jobs exceeding the maximum runtime will be terminated. This run time is an elapsed actual time and not a calculated CPU time (as in LSF). The job flavours are

espresso	= 20 minutes
microcentury	= 1 hour
longlunch	= 2 hours
workday	= 8 hours
tomorrow	= 1 day
testmatch	= 3 days
nextweek	= 1 week

Submit file: `Job Flavours` II

The default job flavour in HTCondor is espresso (20 minutes)[1]. Insted of the fob flavour one could specify directly the number of seconds for job substituting the line with `+JobFlavour` with a line like `+MaxRuntime` = *number_of_seconds*

[1]The python script used to submit FLUKA jobs to HTCondor is instead configured on tomorrow (24 hours) if no other specification is given (see section 4).

Submit file: `queue`

This is the command that schedules the job. It can take an integer as a value (for example `queue 100`) to submit multiple jobs. This means that the same executable specified in the submit file will be submitted multiple times.

Resources and limits

The time limit of a job is specified bu the job flavour or by the maximum run time. By default, a job will get one slot of a CPU core, 2 GB of memory and 20GB of disk space. It is possible to ask for more CPUs or more memory, but the system will scale the number of CPUs you receive to respect the 2GB / core limit. To ask for more CPUs the submit file one can add a line like `RequestCpus = 2`. That case, for example will result in a slot of 2CPUs, 4 GB of memory and 20 GB of disk.

Migration from LFS

Remarks

Interaction with HTCondor system is analogous to LSF but commands are different. Similarly to LSF shorter jobs are more efficient and get scheduled faster. The main differences are:

- standard out and standard error files locations have to be explicitly specified in the submit file
- an LSF job is called cluster in HTCondor. A single simple job is identified in Condor by the cluster number.
- time limits in LSF were in normalized CPU-time relative to an old normalization. In HTCondor they refer to real elapsed time

Submitting FLUKA jobs

Preliminaries

After the migration from LSF to HTCondor, some temporary adjustments were made to the python script `execute.py` written by Tim Cooijman in order to submit jobs to HTCondor in a way similar to what it used to be before the migration from LSF.

The way the FLUKA input file is split with `split.py`, the way the executable `CMSpp` is compiled with `compile.sh` and the way results are combined with `combine.py` are not changed and the syntax for running these programs is the same as before.

Syntax

The syntax for running `execute.py` is now slightly different. For example, assume our FLUKA executable is named `CMSpp`, our input file is named `cmsSTUDY.inp` and we think that each job should be finished in roughly one day. Then the command would be like

```
$ /path/to/execute.py -e CMSpp -q tomorrow cmsSTUDY
```

Essentially the only difference at the moment is that the explicit name of the job flavour has to be written instead of what was the queue name for LSF[2].

[2] The possibility to run locally (`-L`) and submitting jobs for which there is no result present (`--unless-finished`) are still present

Description

To submit jobs to HTCondor the following ingredients are needed: the submit file which will tell HTCondor anything it has to know to submit the job, the executable that HTCondor has to run and folders to organize standard out, standard err and log files for each job.

Each job has its own submit file since each FLUKA input file should have a different random seed. As was mentioned above, using the same submit file with a line like `queue 100` to run multiple jobs, will result in submitting always the same input file.

Description: the submit file

For each input file in the form `cmsSTUDY_*.inp`, `execute.py` will create create a folder `CONDORclustercmsSTUDY_*` and a submit file `submit_cmsSTUDY_*.sub` which could look like this:

```
executable              = CONDORclustercmsSTUDY_aaaa/script_cmsSTUDY_aaaa.sh
output                  = cmsSTUDY_aaaa.$(ClusterId).$(ProcId).out
error                   = cmsSTUDY_aaaa.$(ClusterId).$(ProcId).err
log                     = cmsSTUDY_aaaa.$(ClusterId).$(ProcId).log

universe                = vanilla
+JobFlavour             = "testmatch"
initialdir              = /afs/cern.ch/.../CONDORclustercmsSTUDY_aaaa
transfer_input_files    = /afs/.../cmsv37153_aahr.inp, /afs/.../CMSpp, /afs/cern.ch/.../LBQ-KEK.MAP,
                          /afs/.../cmssw501.fieldmap

queue
```

As it used to be before, the main FLUKA input file, the FLUKA executable and the field maps must be in the directory where `execute.py` is run.

Description: the HTCondor executable

For each input file in the form `cmsSTUDY_*.inp`, `execute.py` will create create a shell script `script_cmsSTUDY_*.sh` which contains all the commands needed to run FLUKA. For example `script_cmsSTUDY_aaaa.sh` will look like:

```
#!/bin/bash
set -e
export LX_ORIGIN=/afs/.../currentdirectory
export LX_INPUT_BASE=cmsSTUDY_aaaa
export LX_INPUT="${LX_INPUT_BASE}.inp"
export LX_EOS=/afs/cern.ch/project/eos/installation/cms/bin/eos.select
export LX_FLOPTS=

[...]

if [[ "CMSpp" ]]
then
  export LX_FLOPTS="-e CMSpp"
fi
export FLUPRO=/afs/.../fluka.2011.2x.4

[...]

$FLUPRO/flutil/rfluka $LX_FLOPTS -M 1 "$LX_INPUT" || true

zip -r "results_${LX_INPUT_BASE}.zip" *"${LX_INPUT_BASE}"* *"${LX_INPUT_BASE}001_fort"* *"ran${LX_INPUT_BASE}"*

mv "results_${LX_INPUT_BASE}.zip" $LX_ORIGIN
rm *${LX_INPUT_BASE}001.log*
rm *${LX_INPUT_BASE}001.out*
rm *${LX_INPUT_BASE}001.err*
rm *"${LX_INPUT_BASE}001_fort"*
rm *"ran${LX_INPUT_BASE}"*
```

Description: `*.log`, `*.err` and `*.out` files

The `cmsSTUDY_*.log` file contains informations about the job (job size, CPUs, memory,...). The `cmsSTUDY_*.err` file contains error messages (if there are any) related to error encountered when submitting the jobs to HTCondor or if the job is terminated. The `cmsSTUDY_*.out` file contains the output of the executable `script_cmsSTUDY_*.sh` and will look like

```
$TARGET_MACHINE = Linux
$FLUPRO = /afs/.../fluka.2011.2x.4

Initial seed copied from /afs/.../fluka.2011.2x.4
Running fluka in /pool/condor/dir_14331/fluka_63
================================ Running FLUKA for cycle # 1 ============================================
Removing links
Removing temporary files
Saving output and random number seed
Saving additional files generated     Moving fort.19 to /pool/condor/dir_14331/cmsv37153_aaaa001_fort.19
     Moving fort.21 to /pool/condor/dir_14331/cmsSTUDY_aaaa001_fort.21

End of FLUKA run  adding: cmsSTUDY_aaaa001.err (deflated 81%)
  adding: cmsSTUDY_aaaa001_fort.19 (deflated 72%)
  adding: cmsSTUDY_aaaa001_fort.21 (deflated 44%)
  adding: cmsSTUDY_aaaa001.log (deflated 60%)
  adding: cmsSTUDY_aaaa001.out (deflated 89%)
  adding: cmsSTUDY_aaaa.inp (deflated 81%)
  adding: rancmsSTUDY_aaaa001 (deflated 46%)
  adding: rancmsSTUDY_aaaa002 (deflated 46%)
```

These files are located in the `CONDORcluster*` folders.

References

Links

Details about the commands and tutorials can be found at
`http://batchdocs.web.cern.ch/batchdocs/index.html`
and at `http://research.cs.wisc.edu/htcondor/manual/`