Job submission and bookkeeping on the UTA farms

Tomasz Wlodek

Abstract: I describe how to submit MC jobs on the Linux farms. This is not a full description of how to operate the farm, just how to submit a MC production run.

Warning: Since each production process is a little bit different, the process of submitting a job is not entirely the same each time. In fact, user has to modify the submission script a little bit for each run. It is therefore important that the user actually *understands* what the job submission scripts do, not merely blindly follow the instructions. Thinking is actively encouraged.

Prerequisities: User should be familiar with Linux. Ability of programming in python is extremely useful.

Step 1. You receive a request from users. Usually requests are listed on page http://www-d0.fnal.gov/computing/mcprod/Requests/Requests.html. You need to know a few things about the process they ask you to generate:

- How many events they request
- What is the generator, what is its version, what are the versions of other executables.
- What is the average number of background events/event requested.
- Where are the generator cardfiles. Which version to use
- What is the detector geometry: mixed or plain
- What is the request number
- Sometimes they give you other, process specific information

What this all means.

Number of events: Usually they request 5000, 10000, 25000, 50000 events. We submit jobs of 500 events each, so a 50000 event request will lead to 100 jobs. We do not generate more than 50000 events per generator job, so if they ask for 200000 event run, we produce four of 50000. (This is because Linux has a 2Gb limit on file size, and a pythia job with more than 50k events would exceed that).

Number of background events: This is number of minimum bias events to be overlayed. Usually 0.5, 1.0 or 2.5.

Generator cardfiles: The generator cardfiles are stored in /mcc-dist/packages/cardfiles/ directory. Look inside this directory. You will see subdirectories containing different cardfile releases. Each of those contains subdirectories containing cardfiles from different physics groups.

Suppose that you are asked to generate process with parent particle z, decay particle ee, for the group np (new phenomena) using cards v00-03-09 and generator pythia. The corresponding cardfiles are /mcc-dist/packages/cardfiles/v00-03-09/np/pythia_z_ee.cards

Alternatively, if you are asked to generate something using cards /mcc-dist/packages/cardfiles/v03- v00-09/top/pythia_ttbar_wtaunu+wtaunu.cards then you know that the cardfile version is v03- v00-09, working group is top, generator is pythia, parent is ttbar and decay is wtaunu+wtaunu.

Step 2. Ok, now when we know all the request details we want to submit generator job. I have a dedicated area on the farm from which I submit jobs. Go to /home/mcfarm/job_creation directory. You will see subdirectories corresponding to various versions of executables: mcp06, mcp06, mcp10 and so on. Go to the subdirectory corresponding to the version of executables you are using now. From now on I will assume that this is mcp10, which is the current release at the time of writing.

The scripts for submitting generator jobs have names of the form make_*parent-decay*_gen, for example make_zprime-ee_gen. You have to prepare a new script each time you submit a new request, use the older ones as templates for current ones.

Let us look carefully at the script now. In the beginning it defines several environment variables (this example is from file make_wjet-wlnu_gen). In the beginning you can see:

```
export GENVERSION=p10.06.01
export RECOVERSION=p10.08.01
export CARDFILEVERSION=v00-03-09
export PARENT=wj
export DECAY=wlnu
export PTMIN=5.
export NBGND=0.5
export NEVT=50000
export PART=a
export REQUEST=1133
export ENERGY=1960.
```

Those variables describe what is the current version of generator code, what is the version of reco code, what is the current version of card files. Then you type in the parent particle and decay particle. The PTMIN variable describes the minimum value of Pt requested. If users do not ask you for other value, use 5. NBGND is the requested number of background events., NEVT% is the requested number of events to be generated.

The PART variable describes which part of generation is this. Usually it is a. When users ask you to produce more than 50000 events of a particular type you have to split the production into several runs. Then the run with first 50000 events has PART=a, the second has PART=b and so on.

REQUEST is simply the request number, which you get from users. ENERGY is the center of mass energy, we set it to 1960. and as far as I can tell we are unlikely ever to change that.

Ok, now let us a bit further into the script. After the declaration of variables you see the following piece of code:

```
cat <<EOF>$LOGFILE
*****************************************************
PROCESS "wjet-->wlnu;pt>$PTMIN"
NEVT    $NEVT
NBGND   $NBGND
PTMIN   $PTMIN
ENCM    1960
REQUEST $REQUEST
-------------------------------------------------------------------------
Log file started at
EOF
date >> $LOGFILE
```

This defines a “run logfile”, a file which will contain information about the production run. This piece of script will initiate the logfile, it will be lated updated and modified automatically.

You have to modify here one line: in the line with PROCESS word, write, between “” signs, a simple description of the process to be generated. This must be human readable and will appear in the bookkeeping table as the description of the process, so that users san find what they want. You can include enviroment variables in this description.

Examples: “Z-- >ee,”, “qcd(udsg)” and so on.

More below in the script you will see a set of mc_runjob steering cards. They are being fed the parameters you described at the beginning of script.

```
cat <<EOF > $SCRIPT

#  Script for starting a pythia-only job
#
#  (Job name will be filled in by reg_job,
#  included here for manual use of fmcreate)
cfg samglobal define string JobName UTA-P-

#  pythia is a configurator for the MCpythia_x program

attach pythia
cfg pythia define string D0Release $GENVERSION
cfg pythia define string RunNumberFile ~/run.number
cfg pythia make seeds
#
cfg pythia define string CardfileVersion $CARDFILEVERSION
cfg pythia define string CardfileDir top
cfg samglobal define string RequestID $REQUEST
#
cfg pythia define int NumRecords 0
cfg pythia define string Production $PARENT
cfg pythia define string Decay $DECAY
cfg pythia define float CollisionEnergy $ENERGY
cfg pythia define float PtGt $PTMIN
cfg pythia define float PtLt 99999.0
cfg pythia define float EtaGt 0.0
cfg pythia define float EtaLt 0.0
```

Sometimes you have to add new cards or remove others, depending on the process you are requested to generate. (This is the point which actually requires thinking on your part).

Here are some examples of what you should add in some typical situations:

If you are requested to produce some top quark decays for a particular value of top quark mass add:
`cfg pythia define float TopMass $TOPMASS` (of course the TOPMASS variable must be defined earlier)

If they want you to generate Higgs boson events with a particular mass of Higgs boson, do:

```
cfg pythia define float HiggsMass $HIGGSMASS
```

If they want to have events in a particular range of Pt, do:

```
cfg pythia define float PtGt $PTMIN
cfg pythia define float PtLt  $PTMAX
```

There are two types of detector geometry: plain and mixed. Mixed is default, but if needed you can change it by adding:

```
attach d0gstar
cfg d0gstar define string Geometry plain
```

If you need to add new mc_runjob cards, study the mc_runjob manual
http://www-clued0.fnal.gov/mc_runjob/

Step 3. Ok, by now you have created your make*gen script. Execute it. The scrupt will create a logfile for your MC production run and submit a generator job.

Let us have a look at the logfile. Its name will be of the form *parent-decay-something-something….*log. The exact form of the log file is determined in the LOGFILE enviroment variable in your make*gen script.

Inside the logfile will look more or less like:

```
PROCESS "wjet-->wlnu"
NEVT    50000
NBGND   0.5
PTMIN   5.
ENCM    1960
```

```
REQUEST 1133
---------------------------------------------------------------------------
Log file started at
Fri Feb  1 14:19:54 CST 2002
Generator job:
UTA-P-wj-wlnu-032142011
Output    : gen_mcp10_p10.06.01_UTA_pythia_topv00+03+09+wj-wlnu-PtGt5.0+PtLt99999.0_mb-
none_1133_032142011
```

What you see here are some data about the MC production you have submitted: number of events, production type, background events and so on. Two new information appeared: the name of generator job (UTA-P-…) and the name of the output file of a generator job (gen_mcp10…)

At this point you have to wait until the generator job is completed and the generator file shows up in the cache. You can inspect status of a job using command:

jobstat *jobname*
if the response is "READY TO DISTRIBUTE" then it is still waiting. If it says "ARCH", it means it has been archived.

Step 4. When the generator job is done, you can submit simulation jobs which will use its output file as input. This is done automatically. Execute the command:

python /home/mcfarm/tomw/start_run.py *your_logfile*

This is a slow script and for big runs it may take a few hours to submit a hundred of jobs.

When completed, you will see that it has appended to your logfile list of names of MC jobs which will use the generator file you have created previously.

Step 5. By now a run has started, the only thing that remains to be done is to move logfile of the MC run to the directory /home/mcfarm/production_logs/runs/current

The bookkeeping package reads all logfiles from this directory and compiles from them a table with production status. The table can be seen from the UTA farm page
http://www-hep.uta.edu/~mcfarm/mcfarm/main.html

Appendix: useful mcfarm commands

jobstat *jobname* : give status of a particular job.
jobstat –r give status of running jobs
jobstat –e, -u, -g :give status of errored, undistributed, gather jobs.
Read jobstat –help for more details
A similar command to jobstat is check_job. You can use it for two purposes: killing a job (check_job –k *jobname)* or fixing a job which has a problem (check_job –f *jobname).*
Again, read check_job –help for details.