ctapipe, the GRID and I

Tino Michael
CEA Saclay, Irfu/SAp

2017-09-19

start off with some useful links
(names are clickable)

DIRC Users Guide how to install dirac client, essential terminal commands and python methods

DIRAC JobMonitor for your jobs submitted by DIRAC; has very limited "resubmit" capabilities, can look at logs, stdout/stderr and output files on-the-fly

MC Prod3 Status description of which MC sets exist and how to obtain them

DIRAC from the command line

I assume you managed to set it up with the User Guide

first, define a bash function for convenience:

```
1  source-dirac (){
2      [[ "$PATH" =~ "dirac/pro/scripts" ]] || \
3          source ~/software/dirac/bashrc;
4      dirac-proxy-init
5  }
```

Retrieve a list of files of your GRID user and save it in a text file in the current directory:
dirac-dms-user-lfns

upload a file to a specifiic storage element (SE):
dirac-dms-add-file /vo.cta.in2p3.fr/user/[initial]/[name]/test.txt test.txt CC-IN2P3-USER

download a file from the SE:
dirac-dms-get-file LFN:/vo.cta.in2p3.fr/user/[initial]/[name]/dir1/job.log

remove a directory and all files within:
dirac-dms-clean-directory /vo.cta.in2p3.fr/user/[initial]/[name]/dir

create a replica on another SE:
dirac-dms-replicate-lfn /vo.cta.in2p3.fr/user/[initial]/[name]/test.txt DESY-ZN-Disk

submitting jobs to the GRID with DIRAC

A very convenient way to submit jobs to the GRID is through a python2 script. All the dirac-specific classes are made available by sourcing the dirac bashrc.
Note: This bashrc will set the python2 (!) version that ships with it as the default *python*, so only source it in a dedicated "submission terminal". It also messes up your *conda* upgrade process. I thought I messed up my installation a few times because of this...

My actual submit script is on ***my github***.

submitting jobs to the GRID with DIRAC

the preamble

```
1 from DIRAC.Core.Base import Script
2 Script.parseCommandLine()
3 from DIRAC.Interfaces.API.Dirac import Dirac
4 from DIRAC.Interfaces.API.Job import Job
5 dirac = Dirac()
```

submitting jobs to the GRID with DIRAC

setting up the input sandbox

sandbox uploads local files to the working directory of your GRID jobs

```
6   input_sandbox = [
7   # files from current directory; will be available
8   # in working directory (WD) during GRID-job
9   "helper_functions.py", "reconstruct.py",
10
11  # file in a different directory; will be in WD (not in sub-dir.)
12  "snippets/append_tables.py",
13
14  # directory in current directory; will be in WD
15  "modules",
16
17  # some directory anywhere on disk; will be (you guessed it) in WD
18  "/local/home/tmichael/software/jeremie_cta/"
19  "sap-cta-data-pipeline/datapipe/",
20
21  # file from a GRID SE (note the LFN prefix); will be in WD
22  "LFN:/vo.cta.in2p3.fr/user/t/tmichael/cta/bin/mr_filter/"
23  "v3_1/mr_filter"]
```

submitting jobs to the GRID with DIRAC

setting up a job object and submitting to the GRID

```
24  for run_filelist in sliding_window(filelist, window_size=10):
25      j = Job()
26      # runtime in seconds times 8 (CPU normalisation factor)
27      j.setCPUTime(6 * 3600 * 8)
28      # set a nice and unique name for your job
29      j.setName('_'.join([channel, mode, run_token]))
30
31      # store results on the SE; supports wildcards in filenames
32      j.setOutputData(["output.log", "output_*.h5"],
33          # `outputPath` relative path starting from your [name]
34          outputSE=None, outputPath="output/path/")
35
36      # files to upload to WD from current machine and other SEs
37      j.setInputSandbox(input_sandbox)

        ...

54      # submit the job to the GRID
55      print('Submission Result: {}\n'.format(
56              dirac.submit(j)['Value']))
```

submitting jobs to the GRID with DIRAC

defining the executable

can be any command available in the WD, local scripts need to be added to the input sandbox to be uploaded

```
38  # takes two strings, first is the executable to run,
39  # second contains additional command line arguments
40  j.setExecutable("reconstruct.py", command_line_arguments)
```

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

naive idea is to ‘setInputData‘ and tell your executable to process them all:

```
38  j.setInputData(run_filelist)
39  j.setExecutable("reconstruct.py", "--input *.simtel.gz")
```

sends the job to the GRID node where files actually are stored

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

naive idea is to ‘setInputData‘ and tell your executable to process them all:

```
38  j.setInputData(run_filelist)
39  j.setExecutable("reconstruct.py", "--input *.simtel.gz")
```

sends the job to the GRID node where files actually are stored, but:

- might be impossible if too many files; no SE has all the data

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

naive idea is to ‘setInputData‘ and tell your executable to process them all:

```
38  j.setInputData(run_filelist)
39  j.setExecutable("reconstruct.py", "--input *.simtel.gz")
```

sends the job to the GRID node where files actually are stored, but:

- might be impossible if too many files; no SE has all the data
- running on only one/few files is not economical
  e.g. there are 45k prod3b-paranal-north-20deg-[gamma,proton] files,
  one file takes 10 min to run (too short),
  produces 45k 100 kB output files (takes three days only to download)

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

or add it to InputSandbox instead:

```
38  j.setInputSandbox(input_sandbox +
39        [input_file_on_SE_1,
40         input_file_on_SE_2,
41         ...])
42  j.setExecutable("reconstruct.py", "--input␣*.simtel.gz")
```

sends your jobs anywhere and pulls data at the beginning to your WD

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

or add it to InputSandbox instead:

```
38  j.setInputSandbox(input_sandbox +
39        [input_file_on_SE_1,
40         input_file_on_SE_2,
41         ...])
42  j.setExecutable("reconstruct.py", "--input␣*.simtel.gz")
```

sends your jobs anywhere and pulls data at the beginning to your WD, but:

- job scheduler does not seem to be aware of "InputSandbox-getting"

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

or add it to InputSandbox instead:

```
38  j.setInputSandbox(input_sandbox +
39        [input_file_on_SE_1,
40         input_file_on_SE_2,
41         ...])
42  j.setExecutable("reconstruct.py", "--input␣*.simtel.gz")
```

sends your jobs anywhere and pulls data at the beginning to your WD, but:

- job scheduler does not seem to be aware of "InputSandbox-getting"
- $\longrightarrow$ if too many jobs try to download files in parallel, interface breaks

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

or add it to InputSandbox instead:

```
38  j.setInputSandbox(input_sandbox +
39        [input_file_on_SE_1,
40         input_file_on_SE_2,
41         ...])
42  j.setExecutable("reconstruct.py", "--input␣*.simtel.gz")
```

sends your jobs anywhere and pulls data at the beginning to your WD, but:

- job scheduler does not seem to be aware of "InputSandbox-getting"
- $\longrightarrow$ if too many jobs try to download files in parallel, interface breaks
- $\longrightarrow$ jobs idle around for 3 h and then get killed

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

or add it to InputSandbox instead:

```
38  j.setInputSandbox(input_sandbox +
39        [input_file_on_SE_1,
40         input_file_on_SE_2,
41         ...])
42  j.setExecutable("reconstruct.py", "--input␣*.simtel.gz")
```

sends your jobs anywhere and pulls data at the beginning to your WD, but:

- job scheduler does not seem to be aware of "InputSandbox-getting"
- $\longrightarrow$ if too many jobs try to download files in parallel, interface breaks
- $\longrightarrow$ jobs idle around for 3 h and then get killed
- local WD has limited size, download too much and you run out of space

submitting jobs to the GRID with DIRAC

defining the input data we want to analyse

what actually works for me

```
38  # leave input sandbox as is
39  j.setInputSandbox(input_sandbox)
40  # loop over all files in the window
41  for run_file in run_filelist:
42      # wait for a randomly up to five minutes before starting
43      sleep = random.randint(0, 5*60)
44      j.setExecutable('sleep', str(sleep))
45      # consecutively downloads the data files,
46      j.setExecutable('dirac-dms-get-file', "LFN:"+run_file)
47      # ... processes them and deletes them again
48      j.setExecutable("reconstruct.py",
49          "--input " + basename(run_file))
50      j.setExecutable('rm', basename(run_file))
51  # merge the output files
52  j.setExecutable("append_tables.py",
53      " ".join(run_filelist)
```

other possibly useful things

```
# if there is a temporary problem at one site, switch it off
j.setBannedSites(['LCG.IN2P3-CC.fr'])

# your large-scale submission was interrupted?
# check which jobs are already there
running_ids = []
for status in ["Waiting", "Running"]:
    running_ids += dirac.selectJobs(status=status,
                                    owner="tmichael")['Value']
running_tokens = []
for id in running_ids:
    # with many jobs, this could take a minute or two
    jobname = dirac.attributes(id)["Value"]["JobName"]
    running_tokens.append(jobname)

# list of available storage elements:
"CC-IN2P3-USER", "DESY-ZN-USER", "CEA-USER", "LAPP-USER",
"CYF-STORM-USER", "CNAF-USER", "LPNHE-USER"
```