Integracion en el cluster con python

Funcion para sbatch

La idea general aqui es utilizar python para enviar tareas al worload manager del cluster (SLURM). Mas o menos el envio con *sbatch* tiene siempre la misma forma asi que lo podemos generalizar a una funcion.

Esta funcion basicamente tendra que escribir un script shell y ejecutarlo. Para ello habria que pasarle determinados parametros que definan la ejecucion que deseamos como el tiempo de ejecucion, la cantidad de cpus o el path de output.

Peor hay un par de cosas mas interesantes que pueden hacerse. Puede definirse si el script a ejecutar tiene algun tipo de dependencia con algun otro que se haya lanzado antes o incluso una dependencia tipo *swarm*.

A ver,

```
  def send_sbatch(env_data):
  """
  This function creates and executes an sbatch script into SLURM

  It takes a dict with the required environment data and returns
  the jobid of the task
  """
  content = '#!/bin/bash\n'
  if 'job_name' in env_data:
    content += '#SBATCH -J '+env_data['job_name']+'\n'
  else:
    content += '#SBATCH -J myjob\n'
  if 'cpus' in env_data:
    content += '#SBATCH -c '+str(env_data['cpus'])+'\n'
    if 'mem_cpu' in env_data:
      content += '#SBATCH --mem-per-cpu='+env_data['mem_cpu']+'\n'
    else:
      content += '#SBATCH --mem-per-cpu=4G\n'
  if 'time' in env_data: content += '#SBATCH --time='+env_data['time']+'\n'
  content += '#SBATCH --mail-user='+os.environ.get('USER')+'\n'
  if 'output' in env_data: content += '#SBATCH -o '+env_data['output']+'-
%j\n'
  if 'partition' in env_data: content += '#SBATCH -p '+partition+'\n'
  if 'command' in env_data:
    content += '#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT\n'
    content += env_data['command']+'\n'
  else:
    content += '#SBATCH --mail-type=END\n'
    content += ':\n'
  if 'filename' in env_data:
```

```
    filename = env_data['filename']
  else:
    filename = 'slurm{:03d}.sh'.format(random.randint(0,1000))
  f = open(filename, 'w')
  f.write(content)
  f.close()
  if 'dependency' in env_data:
    order = ['sbatch --parsable --dependency='+env_data['dependency']+'
'+filename]
  else:
    order = ['sbatch --parsable '+filename]
  return int(subprocess.check_output(order, shell=True))
```

Esta funcion debe recibir toda la informacion dentro de un diccionario que toma por input. Los campos del diccionario, y por ello las variables de ejecucion de sbatch que se toman en cuenta son:

- *filename*: nombre del script que vamos a ejecutar
- *output*: archivo de output de sbatch
- *job_name*: nombre de la tarea
- *cpus*: numero de CPUs que se usaran
- *mem_cpu*: cantidad de memoria por CPU
- *time*: tiempo maximo que se dejara ejecutar la tarea
- *partition*: particion del cluster donde se ejecutara
- *command*: comando a ejecutar (si esta vacio se jecuta el tipico email de aviso)
- *dependency*: lista completa (y/o) tipo de dependencias con otras tareas

Ejemplo: randomise

randomise.py

```
#!/usr/bin/env python

import os
import subprocess
import random

def send_sbatch(env_data):
  """
  This function creates and executes an sbatch script into SLURM

  It takes a dict with the required environment data and returns
  the jobid of the task
  """
  content = '#!/bin/bash\n'
  if 'job_name' in env_data:
    content += '#SBATCH -J '+env_data['job_name']+'\n'
  else:
    content += '#SBATCH -J myjob\n'
  if 'cpus' in env_data:
```

```
    content += '#SBATCH -c '+str(env_data['cpus'])+'\n'
    if 'mem_cpu' in env_data:
      content += '#SBATCH --mem-per-cpu='+env_data['mem_cpu']+'\n'
    else:
      content += '#SBATCH --mem-per-cpu=4G\n'
  if 'time' in env_data: content += '#SBATCH --
time='+env_data['time']+'\n'
  content += '#SBATCH --mail-user='+os.environ.get('USER')+'\n'
  if 'output' in env_data: content += '#SBATCH -o
'+env_data['output']+'-%j\n'
  if 'partition' in env_data: content += '#SBATCH -p '+partition+'\n'
  if 'command' in env_data:
    content += '#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT\n'
    content += env_data['command']+'\n'
  else:
    content += '#SBATCH --mail-type=END\n'
    content += ':\n'
  if 'filename' in env_data:
    filename = env_data['filename']
  else:
    filename = 'slurm{:03d}.sh'.format(random.randint(0,1000))
  f = open(filename, 'w')
  f.write(content)
  f.close()
  if 'dependency' in env_data:
    order = ['sbatch --parsable --dependency='+env_data['dependency']+'
'+filename]
  else:
    order = ['sbatch --parsable '+filename]
  return int(subprocess.check_output(order, shell=True))

# Static covariables
stcov = [6, 7, 8]
# Variables a procesar
dynvars = list(range(9,16))

#Entorno de trabajo
time = '48:0:0'
cpus = 4
wdir = os.environ.get('PWD')

for v in dynvars:
  cdata = {}
  vwdir = wdir+'/var_{:02d}'.format(v)
  if not os.path.isdir(vwdir+'/slurm'): os.mkdir(vwdir+'/slurm')
  cdata['time'] = time
  cdata['cpus'] = cpus
  cdata['job_name'] = 'randomise'
  cdata['filename'] = vwdir+'/randomise_{:02d}.sh'.format(v)
  cdata['output'] = vwdir+'/slurm/randomise_{:02d}-%j.log\n'.format(v)
  cdata['command'] = 'cd '+vwdir+'\n'+'fslmaths '+vwdir+'/stats/GM_merg
```

```
-s 4 '+vwdir+'/stats/GM_merg_s4'+'randomise -i
'+vwdir+'/stats/GM_merg_s4.nii.gz -o '+vwdir+'/stats/fslvbm_{:02d} -m
'.format(v)+vwdir+'/stats/GM_mask -d '+vwdir+'/design.mat -t
'+vwdir+'/design.con -n 5000 -T -V\n'
  send_batch(cdata)
mdata = {}
mdata['job_name'] = 'randomise'
mdata['filename'] = wdir+'/randomise_end.sh'
mdata['output'] = wdir+'/randomise_end.log'
mdata['dependency'] = 'singleton'
send_batch(mdata)
```