CERN-ATS-Note-2018-??? TECH

April 9, 2018

haroon.rafique@cern.ch

PyORBIT on HPC-Batch

Haroon Rafique / BE-ABP

Keywords: PyORBIT, Space Charge, HPC, HPC-Batch, Slurm, PTC, MAD, Python

Summary

A batch HPC cluster using Slurm has been set up for users of MPI applications that cannot run on a single node on the regular batch service. Access is restricted to approved HPC users, mainly from the Accelerator and Technology sector. The system uses the Slurm workload manager.

PyORBIT is well suited to this system. Instructions are provided for installing and running PyORBIT on the HPC-Batch system.

1 Introduction

Official instructions for accessing the HPC-Batch system are available in the knowledge base article KB0004541: How to access and use the batch HPC cluster:

`https://cern.service-now.com/service-portal/article.do?n=KB0004541`

These instructions are repeated and built upon here.

1.1 Hardware

Table 1 shows the available resources on HPC-Batch. There are a total of 217 nodes, each with 40 CPUs, totalling 8680 cores. The nodes are split into four queues, two short (2 day limit), and two long (infinite limit). The ‘batch’ queues may be used by all, and the ‘be’ queues are reserved for members of BE (beams department).

HPC-Batch should not replace HTCondor or standard lxplus simulations, instead it should be used for computationally intense simulations where required.

Queue	Nodes	Time Limit
batch-short	10	2 days (2-00:00:00)
batch-long	66	∞
be-short	69	2 days (2-00:00:00)
be-long	72	∞

Table 1: Computing resources and job time limits on HPC-Batch.

Unfortunately it is not possible to call simulation input files, or in fact executables, from AFS in a reliable manner. Instead one must install PyORBIT on ones HPC-Batch scratch space \hpcscratch \<user>, and copy all simulation data to this local directory. When the simulation is complete, it is recommended that the user copies all important output back to AFS or to EOS, as the scratch space is not backed up. This cannot be done within the submission script, and must be done manually, or in a separate script called by the user (i.e. not called from within the submission script).

1.2 Accessing HPC-Batch

In order to gain access to the HPC-Batch system please request access to the e-group service-hpc-be via e-groups or email `abp-cwg-admin@cern.ch`.

To access HPC-Batch one must log in to the head node `hpc-batch.cern.ch`, and replace <user> with ones CERN account user name:

```
$ ssh -XY <user>@hpc-batch.cern.ch
```

To check the available modules use:

```
$ module avail
```

The system use GCC v4 as default, this is sufficient for PyORBIT, but it can be changed to GCC v6 if required using:

```
$ module load compiler/gcc6
```

MPI versions may also be selected, for PyORBIT, and in general, one must initialise this using:

```
$ module load mpi/mvapich2/2.2
```

This may be included in ones ∼ /.bashrc configuration file, or ones Slurm submission script.

2 Installing PyORBIT on HPC-Batch

There are currently a number of options available when installing PyORBIT. The CERN branches are made available on GitHub by Hannes Bartosik and Jean-Baptiste Lagrange. The original code (developed by SNS) is also available on GitHub. Two options are provided, the first is using Jean-Baptiste Lagrange's versions of PyORBIT and PTC, and the second is replacing the PyORBIT version with one of Hannes Bartosik's branches.

Jean-Baptiste Lagrange's code is located in the public repository:

`https://github.com/jbcern/py-orbit.git`

Hannes Bartosik's code is located in the public repository:

`https://github.com/hannes-bartosik/py-orbit.git`

please note that there are a number of branches; new-features and analytic-space-charge are likely to be the most useful currently.

These branches will all be merged in the near future.

The original PyORBIT code is also available from GitHub, but may not provide the required functionality for CERN users:

`https://github.com/PyORBIT-Collaboration/py-orbit.git`

For tracking, PTC is also required. The latest version to be used for PyORBIT can be found on GitHub:

`https://github.com/jbcern/PTC.git`

The analytical-space-charge-acceleration branch provides the most up-to-date version.

PyORBIT runs within a virtual environment in order to utilise specific versions of libraries such as Python 2.7. As such one must take care of environment variables when running on HPC-Batch.

The user must locally install PyORBIT on HPC-Batch, for this step-by-step instructions are given (if facing difficulties please contact haroon.rafique, or py.orbit at cern.ch):

1. Log into the HPC-Batch head node:

```
        $ ssh -XY <user>@hpc-batch.cern.ch
```

2. Make sure that your `$PATH` and other environment variables are clear in your `~/.bashrc` configuration file.

3. Use the PyORBIT installation script 'install_PyORBIT.sh' given in subsection 4.2. Create this file and copy the contents, making sure to adjust the variables as instructed in subsection 4.2.

4. Create a directory in your home space `\hpcscratch\<user>\`(it is suggested to name this directory `PyORBIT` if you only intend to install a single version of PyORBIT and PTC, otherwise name the directory something else - for example `PyORBIT_new_features` and create a soft link using linux command `ln -s link_name directory_name` called PyORBIT). Enter this directory, copy the install script here, and run it:

```
$ ./install_PyORBIT.sh > install_output.txt &
```

5. The installation script should take a few hours to run. It is difficult to know when it has finished as all output will be pipelined to the `'install_output.txt'` file. Keep checking when this file has been updated, if it is within the last few minutes then the install script is still running.

6. After a few hours check that the install script has finished running, and check the `'install_output.txt'` file in order to ascertain if the installation proceeded with no errors.

7. If no errors are present, it is usually necessary to install PTC manually. Though the install script should have pulled PTC from GitHub, it may not have been installed due to an environment variable in an included macro file. Check the directory `\hpcscratch\<user>\PyORBIT\py-orbit\ext\PTC\\obj`, if there are no `.o` files present, PTC has not been installed.

   In this case run the `make` command in `\hpcscratch\<user>\PyORBIT\py-orbit\ext\PTC\`. It is likely that this will produce the following error:

```
../../conf/make_root_config:5: /conf/make_common_config: No such
   file or directory
../../conf/make_root_config:25: /conf//make_root_config: No such
   file or directory
make: *** No rule to make target '/conf//make_root_config'. Stop.
```

   In order to mitigate this, in the

   `\hpcscratch\<user>\PyORBIT\\py-orbit\conf\make_root_config`

   file, comment out lines 5 and 25 by placing a `#` at the start of each line:

```
include ${ORBIT_ROOT}/conf/${ORBIT_ARCH}/make_root_config
...
include ${ORBIT_ROOT}/conf/make_common_config
```

The intel fortran libraries are required to install PTC, so one must also execute the following source command:

```
$ source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh
```

The Makefile in the `PTC` directory should also be modified:

```
# ------------------------------------------------------------------------------
# External 'include' locations
# ------------------------------------------------------------------------------

INCLUDES += -I/hpcscratch/user/fasvesta/PyORBIT/include/python2.7
```

Now executing the `make` command in \\`hpcscratch`\\<`user`>\\`PyORBIT`\\`py-orbit`\\`ext`\\`PTC`\\shoul be successful. After installing PTC (check the \\`hpcscratch`\\<`user`>\\`PyORBIT`\\`py-orbit`\\`ext`\\`P` for `.o` files as before), remember to uncomment out the lines in the configuration file.

Once these steps are complete, PyORBIT has been installed.

3 Job Submission

Log in to head node (not where things are run).

Using the correct queue

3.1 Slurm Commands

- **squeue** - view job and job step information, -u <user> displays a specific users jobs.
- **sacct** - displays accounting data for all obs and job steps in the Slurm accounting log or Slurm database.
- **sbatch** - submit a batch script to Slurm.
- **srun** - run parallel jobs (used similarly to mpirun).
- **scancel** - used with a job ID to kill the specified job.

3.2 Slurm Script Commands

In a bash script, the following `#SBATCH` commands may be used to specify job parameters:

- **-p** - queue name, for HPC-Batch the options are be-short, be-long, batch-short, batch-long.
- **-N** - number of nodes, each node has 40 cpus on HPC-Batch, and the number of nodes does not need to be specified, only the number of CPUs.
- **-n** - number of cpus.
- **-t** - expected wall time for the job, in the format day-hour:minute:second, for example 1-10:00:00. This is a hard limit, if exceeded your job will be cancelled by Slurm.
- **-o** - the output file created (not from PyORBIT). This file will record what would normally be output to screen/console. A useful format is name.%N.%j.out, where % is the node number, and %j is the Job ID.
- **-e** - the error file created (not from PyORBIT). This file will record what would normally be output to screen/console. A useful format is name.%N.%j.out, where % is the node number, and %j is the Job ID.
- **–mem** - the RAM allocation for this job, for example 10gb.

An example of the use of these variables is shown below:

```
#!/bin/bash
#SBATCH -p be-short
#SBATCH -n 100
#SBATCH --mem 10gb
#SBATCH -t 1-10:00
#SBATCH -o slurm.%N.%j.out
#SBATCH -e slurm.%N.%j.err
```

4 Scripts

A number of scripts are provided as examples or otherwise.

4.1 HPC-Batch Custom Environment

The following environment variable may be used independently, called from within a Slurm submission script, or incorporated into a submission script in order to setup important environment variables.

```
#!/bin/bash
# Use the installed Python version 2.7 (required for PyORBIT)
source /hpcscratch/user/<user>/PyORBIT/virtualenvs/py2.7/bin/activate

# Set up the PyORBIT environment
source /hpcscratch/user/<user>/PyORBIT/py-orbit/customEnvironment.sh

# Intel fortran libraries needed in PTC
source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh
```

4.2 PyORBIT Installation

The install script 'install_PyORBIT.sh' may be used to install PyORBIT on the HPC-Batch cluster. It is provided here but may also be found here:

`\eos\project\p\pyorbit\public\HPC-Batch\`

```
#!/bin/bash

##############################################################################
#   Script to build PTC-pyORBIT from Source with a custom Environment
#   from JB Lagrange Github depositories for pyORBIT and PTC
#   Use of this script:
#   1) create a folder for the whole environment (ex:pyorbit_env)
#   2) copy this script in this folder
#   3) execute this script in this folder
#   After installing everything, you can check things by running the
    examples in py-orbit/examples/
#
#   NB: if you want to recompile pyORBIT after the first installation, you
    need to use:
# cd py-orbit
# source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh
# source customEnvironment.sh
# make clean
# make
##############################################################################

#source ifort for compiling PTC
source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh

#clone pyorbit version from github:
#~ git clone --branch=smooth_binning https://github.com/jbcern/py-orbit.git
git clone --branch=new-features https://github.com/hannes-bartosik/py-orbit.
    git

#clone PTC from github
cd py-orbit/ext
git clone --branch=analytical-space-charge https://github.com/jbcern/PTC.git
```

```
cd PTC
mkdir obj/
cd ../../..

#download and untar sources
echo "download␣and␣untar␣sources..."
curl http://www.mpich.org/static/downloads/3.2/mpich-3.2.tar.gz | tar xvz
curl https://www.python.org/ftp/python/2.7.12/Python-2.7.12.tgz | tar xvz
curl http://zlib.net/fossils/zlib-1.2.11.tar.gz | tar xvz
curl http://www.fftw.org/fftw-3.3.5.tar.gz | tar xvz
curl https://pypi.python.org/packages/source/v/virtualenv/virtualenv-15.0.0.
    tar.gz  | tar xvz

#build python
echo "build␣python2.7..."
cd Python-2.7.12
./configure -prefix=`pwd`/..
make
make install
cd ..

#build zlib
echo "build␣zlib..."
cd zlib-1.2.11
./configure -prefix=`pwd`/..
make
make install
cd ..

#build mpi
echo "build␣mpich..."
cd mpich-3.2
./configure -prefix=`pwd`/.. --disable-fortran
make
make install
cd ..

#build fftw
echo "build␣fftw..."
cd fftw-3.3.5
./configure -prefix=`pwd`/.. --disable-fortran --enable-mpi MPICC=`pwd`/../
    bin/mpicc
make
make install
cd ..

#build python packages
echo "build␣python␣packages..."
source py-orbit/customEnvironment.sh
cd virtualenv-15.0.0
../bin/python setup.py install

cd ..
mkdir virtualenvs
```

```
cd virtualenvs
../bin/virtualenv py2.7 --python=../bin/python
cd py2.7/bin
source activate

#Add here the python packages you want to install
echo "installing numpy..."
./pip install numpy
echo "installing scipy..."
./pip install scipy
echo "installing ipython..."
./pip install ipython
echo "installing matplotlib..."
./pip install matplotlib
echo "installing h5py..."
./pip install h5py
echo "DONE"
echo
cd ../../..

#build pyorbit
echo "Building pyORBIT..."
cd py-orbit
source customEnvironment.sh
make clean
make
```

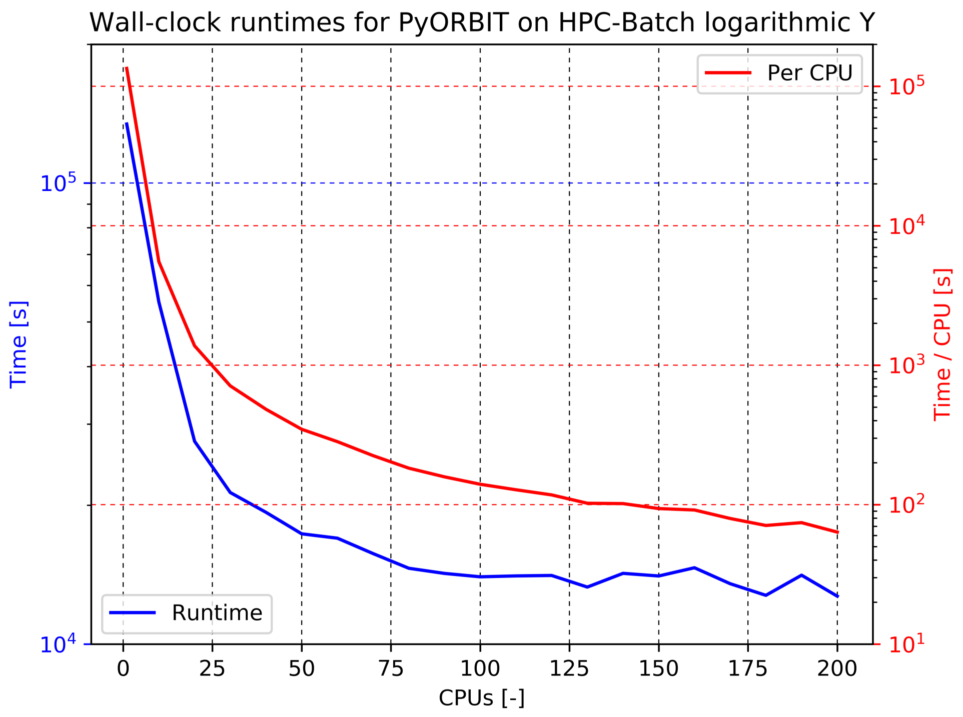

Figure 1: Run time per CPU for the test case.

5 Run Times

Standard test case 3000 macro particles, 10^5 turns, PS, frozen space charge.

Peak speedup at around 30 macro-particles per core. Expect due to bottleneck as number of macro particles is small.

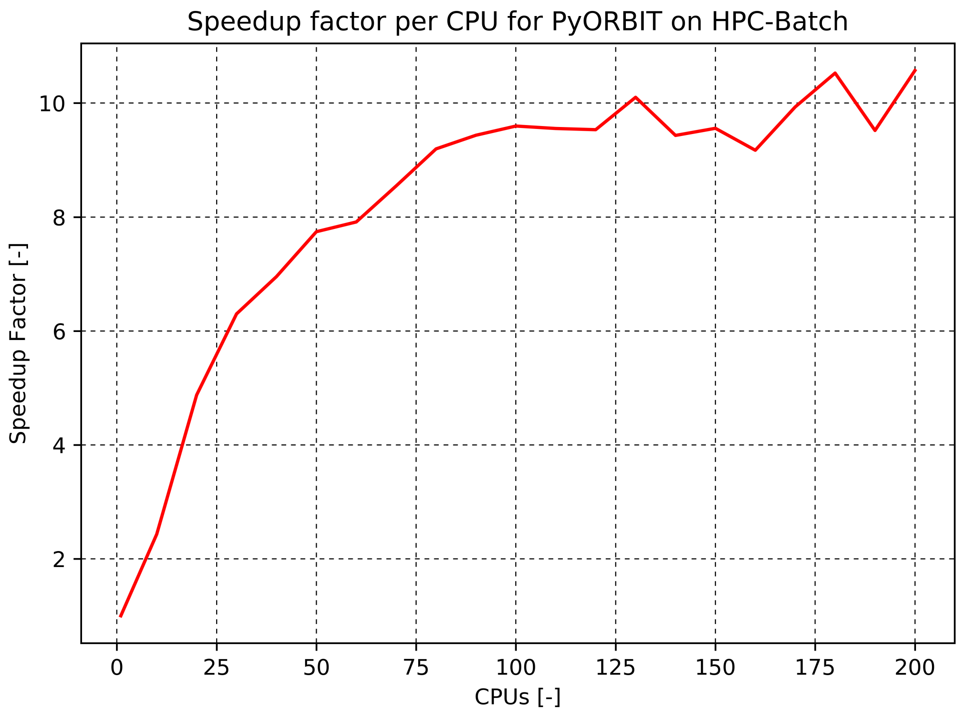

Figure 2: Speedup for the test case.