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

April 18, 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 the Slurm (Simple Linux Utility for Resource Management) workload manager has been set up for users of MPI (Message Passing Interface) applications that are massively parallel and computationally intensive. Access is restricted to approved HPC (High Performance Computing) users, mainly from the Accelerator and Technology sector.

PyORBIT, a Python/C++ implementation of the ORBIT (Objective Ring Beam Injection and Tracking) code used at CERN for space charge simulations, 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 expanded upon here.

1.1 Hardware

Table 1 shows the available resources on HPC-Batch. There are a total of 217 nodes, each with 20 physical CPUs, and each viewed as 2 cores due to hyperthreading, totalling 8680 'cores' (or 4340 physical 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).

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.

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

Unfortunately it is non-trivial 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`, replacing <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. After being granted access, log into the HPC-Batch head node:

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

2. Make sure that your `$PATH` and other environment variables in your `~/.bashrc` configuration file do not conflict with those required for PyORBIT, for example the `$PYTHONPATH` should be empty.

3. Use the PyORBIT installation script 'install_PyORBIT.sh' given in subsection 4.3, making sure to adjust the variables as instructed in subsection 4.3. It can be copied from eos:

```
$ cp eos/project/p/pyorbit/public/HPC-Batch/install_PyORBIT.sh .
```

4. Create a directory in your home space \hpcscratch\<user>\. It is strongly suggested to name this directory `PyORBIT` if you only intend to install a single version of PyORBIT and PTC. If you intend to use multiple versions then name the directory something else - for example `PyORBIT_new_features` and create a soft link using linux command `ln -s directory_name link_name` called PyORBIT, for example:

```
$ mkdir PyORBIT_new_features
$ ln -s PyORBIT_new_features PyORBIT
```

   It is important that the name of this folder is not changed after installation, as certain virtual environment variables are set using the folder name at installation. Enter the PyORBIT directory, copy the install script here, and run it:

```
$ mkdir PyORBIT
$ cd PyORBIT
$ cp /eos/project/p/pyorbit/public/HPC-Batch/install_PyORBIT.sh .
$ ./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 conflict. Check the directory \hpcscratch\<user>\PyORBIT\py-orbit 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/<user>/PyORBIT/include/python2.7
```

Now executing the `make` command in

`\hpcscratch\<user>\PyORBIT\py-orbit\ext\PTC\`

should be successful. After installing PTC (check the

`\hpcscratch\<user>\PyORBIT\py-orbit\ext\PTC\obj\`

directory for `.o` files as before to confirm the installation), remember to uncomment out the lines in the configuration file.

Once these steps are complete, PyORBIT has been installed.

3 Job Submission

1. Log into the HPC-Batch head node:

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

2. Using the `sinfo` command, check availability in the queues (see Fig. 2 for an example), and choose the appropriate queue for your submission. If your job should take less than 2 days, use one of the short queues, otherwise use a long queue. The 'be' queues are reserved for members of the beams department at CERN, the 'batch' queues may be used by any member of CERN.

3. Using the `#BATCH` commands (see Section 3.2), create a submission script in bash. For an example of this see Section 4.2.

4. You must copy all files related to the job to the local `\hpcscratch\<user>` directory. HPC-Batch is part of the AFS and EOS filesystems, though these may not (in general) be used in a submission script. Any file transfers to or from AFS or EOS must be done outside of the job (i.e. the submission script or executable).

5. Submit the job(s) using the `sbatch` command:

```
$ sbatch submission_script.sh
```

```
14:05:01 harafiqu slurmgate04 /hpcscratch/user/harafiqu
→ sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch-short    up 2-00:00:00     10   idle hpc[004-013]
batch-long     up   infinite      1   comp hpc051
batch-long     up   infinite      1    mix hpc018
batch-long     up   infinite     27  alloc hpc[014-017,019-033
batch-long     up   infinite     37   idle hpc[052-059,061,065
be-short       up 2-00:00:00      2  drain hpc-be[021,023]
be-short       up 2-00:00:00      1    mix hpc-be008
be-short       up 2-00:00:00     43  alloc hpc-be[001-002,006-
3-124,126,128,134-136,139]
be-short       up 2-00:00:00     23   idle hpc-be[012-013,016,
be-long        up   infinite     36  alloc hpc-be[003-005,009,
0-111,119]
be-long        up   infinite     36   idle hpc-be[010,029,048,
3,137-138]
```

Figure 1: `sinfo` command output on HPC-Batch.

6. You may use Slurm commands (shown in Section 3.1) to observe the progress of your submission.
7. When your job is complete, it is suggested that you move the output and important data to AFS or EOS. Any file transfers to or from AFS or EOS must be done outside of the job (i.e. the submission script or executable).

3.1 Slurm Commands

The following Slurm commands may be useful when using HPC-Batch, they are described in detail on the Slurm manual pages: https://slurm.schedmd.com/man_index.html

- **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.
- **sinfo** - displays the status of all available nodes, an example of this command on the HPC-Batch system is given in Fig. 2.

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 threads over 20 cpus on HPC-Batch, and the number of nodes does not need to be specified, only the number of tasks.
- **-n** - number of tasks (threads).
- –**job-name** - job name.
- **-t** - wall-clock time limit 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 %N 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 %N is the node number, and %j is the Job ID.
- –**mem** - the RAM allocation for this job, for example 12gb.

For the HPC-Batch system, it is necessary to specify a queue `-p`, the number of tasks (threads) required `-n`, and the time limit `-t`. An example use of these variables is shown below:

```
#!/bin/bash
#SBATCH -p be-short
#SBATCH -n 123
#SBATCH --mem 12gb
#SBATCH --job-name TestJob
#SBATCH -t 1-23:45:01
#SBATCH -o slurm.%N.%j.out
#SBATCH -e slurm.%N.%j.err
```

3.3 Possible Job Errors

A common cause of errors when using PyORBIT on the HPC-Batch system is the improper creation of the virtual environment on each node.

When encountering Python errors such as `ImportError: No module named ...`, it is likely that either the node is not running the correct version of Python, or the `$PYTHONPATH` is not correctly set. Try outputting these and making sure that they correspond to the correct directory in your PyORBIT installation. The `which python` command in a terminal will show if the system Python version is running. One must activate Python 2.7 if using PyORBIT, this is shown in Fig **??**.

When encountering Fortran errors, the most likely candidate is a lack of sourcing the intel Fortran libraries that are required in PTC. This should be rectified using the following

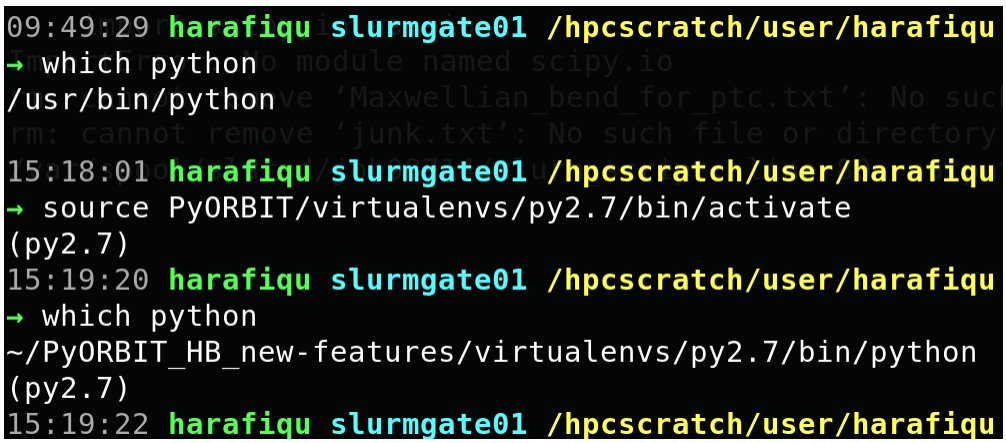

Figure 2: `which python` command output on HPC-Batch, and activation of Python 2.7, which is installed as a prerequisite for PyORBIT.

command:

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

PyORBIT uses MPI for parallelisation, and this is done rigorously, however a user script may not properly take into account MPI concerns. For example if all threads/cores need to interact, they must all be at the same stage in the simulation. Or if one thread is creating a file that is needed by all threads, the others must wait until the file is created before reading it. When encountering MPI errors, it is suggested that these synchronisations are handled correctly.

4 Scripts

A number of scripts are provided as examples or for reference.

4.1 HPC-Batch Custom Environment

The following environment variables may be used independently, called from within a Slurm submission script, or incorporated into a submission script.

```
#!/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 virtual environment
source /hpcscratch/user/<user>/PyORBIT/py-orbit/customEnvironment.sh

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

4.2 Sbatch Job Submission Script

The following bash script is an example of a Slurm batch submission script. It is run with the command:

```
$ sbatch submission_script.sh
```

as described in section 3. Using the commands detailed in section 3.2 one can setup the parameters of ones batch job on the HPC-Batch system.

```
#!/bin/bash
#SBATCH -p batch-short
#SBATCH -n 100
#SBATCH -t 1-23:59
#SBATCH --job-name TestJob
#SBATCH -o slurm.%N.%j.out
#SBATCH -e slurm.%N.%j.err

# load the mpi module we wish to use
module load mpi/mvapich2/2.2

# Tell the system where to find the extra libraries we have installed
# $ORBIT_ROOT will be set in the customEnvironment.sh
export PYTHONPATH=${PYTHONPATH}:${ORBIT_ROOT}/py:${ORBIT_ROOT}/lib:${
    ORBIT_ROOT}/virtualenvs/py2.7/lib/python2.7/site-packages/

# source intel libraries and custom environment variables
source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh
source /hpcscratch/user/<user>/PyORBIT/py-orbit/customEnvironment.sh

# We need to activate the virtual python environment
VIRT_PY_DIR='/hpcscratch/user/<user>/PyORBIT/virtualenvs/py2.7/bin'
cd ${VIRT_PY_DIR}
source activate

# We redefine ORBIT_ROOT in this closed environment instead of taking it from
    the system
# This is optional
ORBIT_ROOT='/hpcscratch/user/<user>/PyORBIT/py-orbit'

# This is the root working directory on HPC-Batch
BATCH_ROOT_DIR='/hpcscratch/user/<user>'

# This is the directory of ones PyORBIT simulation
RUN_DIR='/hpcscratch/user/<user>/PyORBIT/py-orbit/examples/
    CERN_PSB_spacecharge'

# OPTIONAL: Create a simulation output file to save runtime and other info
cd ${BATCH_ROOT_DIR}
output_dir='output'
```

```
mkdir -p $output_dir
simulation_info_file="${BATCH_ROOT_DIR}/${output_dir}/simulation_info_${
    SLURM_JOB_ID}.${SLURM_NODEID}.${SLURM_PROCID}.txt"

echo "PyOrbit path:  `readlink -f ${ORBIT_ROOT}`" >> ${simulation_info_file}
echo "Run path:  `readlink -f ${RUN_DIR}`" >> ${simulation_info_file}
echo "Submit host:  `readlink -f ${SLURM_SUBMIT_HOST}`" >> ${
    simulation_info_file}
echo "SLURM Job name:  `readlink -f ${SLURM_JOB_NAME}`" >> ${
    simulation_info_file}
echo "SLURM Job ID:  `readlink -f ${SLURM_JOB_ID}`" >> ${simulation_info_file}
echo "SLURM Nodes allocated:  `readlink -f ${SLURM_JOB_NUM_NODES}`" >> ${
    simulation_info_file}
echo "SLURM CPUS per Node:  `readlink -f ${SLURM_CPUS_ON_NODE}`" >> ${
    simulation_info_file}
echo "SLURM Node ID:  `readlink -f ${SLURM_NODEID}`" >> ${simulation_info_file
    }
echo "SLURM total cores for job:  `readlink -f ${SLURM_NTASKS}`" >> ${
    simulation_info_file}
echo "SLURM process ID:  `readlink -f ${SLURM_PROCID}`" >> ${
    simulation_info_file}
echo "**************************************" >> ${simulation_info_file}

# Now run the executable - first enter the directory
cd ${RUN_DIR}

# start timer
tstart=$(date +%s)

# Normally PTC PyORBIT would be run with -np (number of cores):
# $MPIBIN/mpirun -np $nnodes ${ORBIT_ROOT}/bin/pyORBIT ${RUN_DIR}/pyOrbit.py
# However this is specified at the start of this file with the -n SBATCH
    command
# Therefore we run 'srun' like mpirun, with no -np argument:
srun ${ORBIT_ROOT}/bin/pyORBIT ${RUN_DIR}/pyOrbit_simulation_file.py

# Some example PTC cleanup
rm Maxwellian_bend_for_ptc.txt
rm junk.txt

#end timer and add to simulation_info file
tend=$(date +%s)
dt=$(($tend - $tstart))
echo 'total simulation time (s): ' $dt >> ${simulation_info_file}
```

4.3 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 in the PyORBIT EOS directory:

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

The lines beginning `git clone` may be modified if a different version of PyORBIT or PTC is required. An example is given at line 22 where the new-features branch of Hannes Bartosik's PyORBIT git repository is selected, and the smooth_binning branch of Jean-Baptiste Lagrange's git repository is commented out.

```
#!/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
```