Job Submitter Documentation

Release 0+untagged.133.g5a1e521.dirty

Juan Eiros

February 27, 2017

Contents

Contents:

CHAPTER 1

Job Submitter

I've started developing this Python program to generate the appropriate files for long classic Molecular Dynamics (MD) runs in the Imperial College HPC facility, using the AMBER MD engine (GPU version).

The objective is to automate the process, so you can chain several jobs and get the results of each one directly to your machine. No more manual edit of your submission scripts, copying restart files back and forth, etc. All is needed is to specify the settings of your simulation in a configuration JSON file and then chain the PBS jobs using dependency on each other.

Maybe this can be useful for other people as well, I think this should be fairly general for other HPC facilities.

Before you start

You need to set up your passwordless ssh from your local machine to the HPC. To test if it works properly, you should be able to `scp` a file from the HPC to your local machine and not be prompted for your password. Like so:

```
$ scp username@HPC-hostname:/home/username/test_file.txt .
test_file.txt                100%    0      0.0KB/s   00:00
```

You should also check that rsync is available in your HPC cluster, since it is used to transfer the files (should be available in any Linux distribution, I think).

Create an example input file using the *jobsubmitter example* command.

- Free software: MIT license
- Documentation: https://JobSubmitter.readthedocs.io.

Features

- TODO

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

From sources

The sources for Job Submitter can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/jeiros/JobSubmitter
```

Or download the tarball:

```
$ curl  -OL https://github.com/jeiros/JobSubmitter/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Usage

Basic workflow

Select your settings in the JSON file. There is an example file `input_example.json`. You can also generate an input example file with `JobSubmitter skeleton`.

The code uses Python 3. Test your Python version in your machine with `python --version`.

Then, use the program with:

```
JobSubmitter generate_scripts input_example.json
```

This will generate a series of `.pbs` files that have to be copied to the HPC along with the `launch_PBS_jobs` script and the appropriate files to run the MD job (topology, any restart/inpcrd files, as well as the input files with the MD settings.)

Once you're in the appropriate HPC directory (the one you've specified in the `job_directory` variable), submit the jobs with `launch_PBS_jobs`.

JSON inputs

Scheduler

At the moment only the PBS job scheduler is implemented.

HPC_job

Set this to `true` if the job is going to be run in the HPC at Imperial. Set to `false` of leave empty if you want to run on a local machine that has torque installed.

PBS settings

These settings are used to build the PBS directives as headers.

- `walltime` Specify the walltime to be used in format `hh:m:s`.
- `nnodes` Nodes to be used.
- `ncpus` Number of cores to be used.

- `ngpus` Number of GPU cards to be used.
- `mem` Specify the memory (in MB) for the job in the format `XXXXmb`.
- `host` Host were the job is going to run.
- `queue` Queue were the job is going to run. Only two options are supported for the HPC:
  - `qpgpu` for public chemistry department queue
  - `pqigould` for the private queue.

To run on the local machines, there is the 'long' queue with a walltime of 192 hours enabled.

- `gpu_type` The type of GPU to be used.

Simulation details

- `system_name` The name of your system. This is used throughout the code to give the files matching names.
- `inpcrd_file` The input coordinates file. This is used if you want to start your simulation from 0. Should end with `.inpcrd`.
- `topology_file` The topology file of your system. Should end with `.prmtop`
- `start_rst` The restart file that the first job is going to use. If you start from 0 and want to run pre-production commands in the GPU (discouraged), this should match the name of the restart file that is written after your last pre-production run (usually a heating protocol). If you don't start from 0, this file will be read to start the first job.
- `input_file` The input file with the MD settings for the production run. You should be specially careful that the timestep (`dt`) and number of MD steps to be performed (`nstlim`) match the `job_lenght` that you want, as the program does not do this for you nor checks if it is correct.
- `start_time` The time from which you want to launch the simulation (in nanoseconds). Doesn't necessarily have to be 0 (you can start from an existing simulation, using the appropriate `.rst` file, as specified in the `start_rst` variable.)
- `final_time` The time at which you want your simulation to stop (in nanoseconds).
- `job_length` The lenght of each individual MD run (in nanoseconds). You should set accordingly the amount of MD iterations and timestep to be used in your MD input file. Also, be careful not to hit the wallclock time.
- `job_directory` The directory in which the job is going to be run in the HPC. You should launch the `launch_PBS_jobs` script from here once all the necessary files are in it. This is the directory were all the `.pbs` & the rest of the input files should be. Also, this is where you issue the `launch_PBS_jobs` command.
- `cuda_version` The cuda version to use via `module load cuda`. This is expected to not changed very frequently.
- `binary_location` The full path to the `pmemd.cuda_SPFP` binary (or whatever it's called). This is expected to not changed very frequently.
- `pre_simulation_cmd` An indefinite list of commands that you want to run before the production run. These can be run on the HPC or locally. Nothing is assumed here, they'll be run as is (so if you want them to run in the HPC the binary location should match the one in the HPC, for instance).
- `pre_simulation_type` Where to run the pre-production commands. Two options are supported:
  - `cpu`: Whatever commands you want to run before the production run are read from the `pre_simulation_cmd` section in the JSON file and are written to a bash script called `pre_simulation.sh` which you can then run in your machine.

- `gpu`: If you want to run the *pre_simulation_cmd* commands in the HPC. Then they will be used in the first `.pbs` file. This is not recommended as for some systems GPUs are known to give trouble with minimisations.

Local Machine

- `user` Your username in your local machine. Find it with the `whoami` command.
- `hostname` The hostname of your machine. Find it with the `hostname` command.
- `destination` The *full path* in which the results of the simulations are going to be moved to. This directory should exist before the data copy is attempted, or else it will fail.

Master Node

This is just used if the jobs are run on the local machines.

- `user_m` Your username on the master node.
- `hostname_m` The hostname of the master node. Shouldn't change.
- `job_directory_m` The job where you'll launch the `.pbs` scripts from.

Example JSON input file

Here's an input file to get you started:

```
{
    "scheduler": "pbs",
    "HPC_job": "True",
    "pbs_settings": {
        "walltime": "72:0:0",
        "nnodes": 1,
        "ncpus": 1,
        "ngpus": 1,
        "mem":"1000mb",
        "host":"cx1-51-6-1",
        "queue":"gpgpu",
        "gpu_type":"K80"
    },
    "simulation_details": {
        "system_name": "protein1",
        "inpcrd_file": "protein1.inpcrd",
        "topology_file": "protein1.prmtop",
        "start_rst": "Heated_eq.rst",
        "input_file": "Production_cmds.in",
        "start_time": 0,
        "final_time": 500,
        "job_length": 50,
        "job_directory": "/work/username/protein1",
        "cuda_version": "7.5.18",
        "binary_location": "/path/to/AMBERHOME/bin/pmemd.cuda_SPFP",
        "pre_simulation_cmd": [
            "/path/to/AMBERHOME/bin/pmemd.cuda_SPFP -O -i premin.in -o premin.out -c ${inpcrd} -p ${p
            "/path/to/AMBERHOME/bin/pmemd.cuda_SPFP -O -i sandermin1.in -o sandermin1.out -c premin.r
            "/path/to/AMBERHOME/bin/pmemd.cuda_SPFP -O -i 02_Heat.in -o 02_Heat.out -c sandermin1.rst
            "/path/to/AMBERHOME/bin/pmemd.cuda_SPFP -O -i 03_Heat2.in -o 03_Heat2.out -c 02_Heat.rst
        ],
        "pre_simulation_type": "gpu"
    },
    "local_machine": {
        "user": "username",
        "hostname": "hostname",
        "destination" : "/Users/username/protein1"
    },
    "master_node": {
        "user_m": "username",
        "hostname_m": "master_node-hostname",
        "job_directory_m": "/home/username/protein1"
    }
}
```

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/jeiros/JobSubmitter/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

Job Submitter could always use more documentation, whether as part of the official Job Submitter docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jeiros/JobSubmitter/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *JobSubmitter* for local development.

1. Fork the *JobSubmitter* repo on GitHub.
2. Clone your fork locally:

```
    $ git clone git@github.com:your_name_here/JobSubmitter.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
    $ mkvirtualenv JobSubmitter
    $ cd JobSubmitter/
    $ python setup.py develop
```

4. Create a branch for local development:

```
    $ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
    $ flake8 JobSubmitter tests
    $ python setup.py test or py.test
    $ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
    $ git add .
    $ git commit -m "Your detailed description of your changes."
    $ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/jeiros/JobSubmitter/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_JobSubmitter
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search