Using the Submitter class for high throughput HPC calculations with Soprano

Simone Sturniolo

November 4, 2016

Chapter 1

The Submitter class

In Soprano, the hpc.submitter module provides classes and tools meant to help computational scientists to automate the process of generating, running and possibly post-processing huge batches of molecular calculations like molecular dynamics, DFT calculations and so on on high performance computing machines where access to the nodes is regulated by a queueing system. The target is to allow users to create their own pipelines for high throughput calculations, like running the same analysis tools through an entire database of chemical structures, with only a few lines of Python code. Since this is, however, still a complex task that will inevitably require to be customised and tailored to the needs of each specific case, using this functionality requires a somewhat higher level of programming confidence than the rest of the library and can look a little daunting at first. No worries - this manual will guide you through the steps necessary to deploy your own Submitter to your machine of interest to run potentially any computational task you can think of. Besides instructions on the creation of a new Submitter, this manual also will include advice on how to test it and debug it locally, in order to minimise the risk of running into bugs after deploying the system.
First, let us clarify what a Soprano Submitter instance *is* and *isn't*. A Submitter is a very general class designed to automate the process of queueing, running and postprocessing huge amounts of calculations on a computing cluster. It is *not*:

- a system designed to push calculations to a remote machine. A Soprano Submitter is meant to be run on the same machine where the calculations are going to be run;
- a system handling parallel calculations. There is no parallelism included in a Submitter. As the name suggests, it simply submits jobs to a queueing system like GridEngine, which then itself provides them with the required number of cores. The Submitter is entirely agnostic of the underlying hardware architecture;

- a system ready for deployment as-is. As it comes shipped with Soprano, the Submitter class basically does nothing. It is meant to be used as a parent class to write your own sub-class which specifies the details of the task of interest through a number of designated methods. This is absolutely on purpose - the objective here is maximum flexibility for potentially very demanding tasks, sacrificing immediate usability.

Now let's see a quick overview of what the typical pipeline for deploying a Submitter system is, followed by a more detailed explanation of all the steps involved.

Chapter 2

Deploying a Submitter

The process to create and launch a Submitter requires the following steps:

- Write a class inheriting from soprano.hpc.submitter.Submitter implementing the desired calculation through its core methods;
- Write a Python source file creating an instance of this class (it can also be the same file) and setting up its parameters, and positioning this file in the working directory. In some cases this might require creating a soprano.hpc.submitter.QueueInterface as well;
- Create a submission script for your HPC system using tags which are going to be replaced by specific values in each job;
- Launch the newly created submitter from the command line by calling the appropriate Soprano script.

Let's see these steps in more detail.

2.1 Writing a Submitter class

The basic template to inherit from the Submitter class can be as simple as this:

```
from soprano.hpc.submitter import Submitter

class MySubmitter(Submitter):
    pass
```

However, such a class would be of no use since it does not implement any particular behaviour. The Submitter class originally has a number of methods. Some of them are preceded by an underscore and are meant to not be touched, as they handle the core behaviour of any Submitter. Others are meant to be overridden when inheriting from the base class and control custom behaviour -

the specifics of the problem at hand. The core methods determine the Submitter workflow, which can be described as follows: