FHI-aims and pymatgen, atomate, fireworks and custodian – High Throughput made simple –

Jan Kloppenburg

September 5, 2019

1 What is it?

Python Materials Genomics (pymatgen) is a robust materials analysis code that defines core object representations for structures and molecules with support for many electronic structure codes. It is currently the core analysis code powering the Materials Project, see `https://github.com/materialsproject/pymatgen` and `https://materialsproject.org/`

2 What's new?

I have written an import and export interface to adopt the standard `geometry.in` format from FHI-aims to be read and written by pymatgen. Additionally there are now simple workflows available for FHI-aims in the atomate package. Atomate provides workflows for VASP structure relaxations, GW and many other computations with appropriate validators. The current FHI-aims workflows are for structure relaxations with validators for convergence and a sample workflow for TDDFT computations also with convergence validation and additional validation of created file output from the TDDFT routine. Validators and routines checking the current computation are implemented in the custodian package, see `https://github.com/materialsproject/custodian`. These workflows are usually wrapped within the fireworks framework. Fireworks (`https://github.com/materialsproject/fireworks`) provides a database linked platform for distributed computations on different compute clusters. The actual cluster configuration is not important for the definition of a workflow. It only needs to be defined on the working cluster.

3 How does is work in practice?

As an example I want to show the outline of my testing set. I chose 1000 random molecules from a testing database. For each of these I did the following:

1. light relaxation
2. tight relaxation
3. TDDFT singlet computation

Each structure relaxation is validated for convergence. If this validation fails the respective firework will be marked in state FIZZLED and can be restarted or modified before restart. Each single item in the list is one firework. Bundling these three together forms one workflow. The dependencies can be set arbitrarily, in this test case they are $1 \rightarrow 2 \rightarrow 3$. This means firework 3 can not start before 2 and 1 have successfully finished. For practical demonstration we have now created a database that contains

1000 workflows, each of which contains three fireworks totalling 3000 computations. The script execution on a laptop takes about one minute to stuff the database. The validation is done by custodian which is the babysitter for FHI-aims. It stems from VASP and who has run VASP knows that it can really make good use of a babysitter. FHI-aims does not have many instabilities but in principle any possible error (i.e. force $\leftrightarrow$ energy inconsistency) can be detected during runtime and handled accordingly without the need to resubmit the job. Custodian runs FHI-aims, it can modify the `control.in` and `geometry.in` depending on the error and then restart the computation.

Furthermore, one can define detours within workflows or `if`-conditions, i.e. during random mixing of alloys and a following structure relaxation we can check whether the system is metallic and then run the appropriate computation after this check.

3.1 Cluster setup

To run the actual computations we use two different clusters as an example. Cluster A has 16-core nodes and cluster B has 24-core nodes. On cluster A the configuration for a worker can be set in different ways:

- One worker occupying a single node with 16 cores, submitting 10 workers results in 10 jobs in the queue, totalling 160 cores
- One worker occupying 160 cores using 16 cores per job but resuling in a single job in queue

For cluster B we can use an appropriate configuration for 24 nodes. Each single worker will contact the central database and request a job to work on. When done the respective firework state will be marked and a next job is requested. Workers automatically termintate when the database contains no more fireworks of state READY. In this scenario I do not have to care for any priority in the queue on any cluster because I don't need to manage my jobs manually. That is I do not have to physically move the input files from a cluster to another because the queue is empty. I just submit the workers where I have priority.

4 Getting the modified codes

I have not made any pull requests into the mainline of the codes yet. The modifications concern custodian, pymatgen and atomate. These codes are available from clones of the original github repositories and publicly available at:

1. pymatgen: `https://github.com/janklinux/pymatgen`
2. atomate: `https://github.com/janklinux/atomate`
3. custodian: `https://github.com/janklinux/custodian`

The FHI-aims interfaced cluster expansion code CASM is also available at `https://github.com/janklinux/CASMcode`.

4.1 Example usage

I could provide example scripts and tutorials on interest. Python3.6+ and a running mongodb server are required. The setup of a mongodb instance is straight forward and can be used on a laptop for testing. Usually the code stuffing the database consists of a few lines of python. New fireworks and respective validators can be implemented easily in custodian and atomate. I would be happy to integrate examples into a workshop with talk or hands-on tutorials.