The Journal of Open Source Software

MyQueue: Task and workflow scheduling system

Jens Jørgen Mortensen[1], **Morten Gjerding**[1], **and Kristian Sommer Thygesen**[1]

1 CAMD, Department of Physics, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

DOI: 10.21105/joss.01844

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @gonsie
- @marksantcroos

Submitted: 29 October 2019
Published: 14 January 2020

Summary

Task scheduling and workload management on high-performance computing environments is usually done with tools such as `SLURM` (Jette, Yoo, & Grondona, 2002). MyQueue is a front-end for schedulers that makes handling of tasks easy. It has a command-line interface called *mq* with a number of sub-commands and a Python interface for managing workflows. Currently, the following schedulers are supported: SLURM, PBS, and LSF.

The idea behind `MyQueue` is to define a personal queue that the user can interact with in an easy and efficient way while `MyQueue` handles the interaction with the scheduler. Finished tasks will stay in the personal queue until they are explicitly removed so they can be listed with their status (done, failed, timed-out, or out-of-memory). This makes it easy to keep track of your tasks: If a task is listed as "done", it reminds you that some action should be taken, e.g., the result of the task should be checked. If a task failed then you need to fix something and resubmit the task. In this sense, `MyQueue` works as a to-do list.

`MyQueue` has a convenient *list* sub-command. It will by default only show tasks belonging to the current folder and its sub-folders, making it easy to manage several projects by putting them in separate folders. Failed tasks will show a short error message read from the relevant line in the error file. You can select the tasks you want to list by status, task-id, name, or error message. A task can be marked with a *restarts* number N, indicating that `MyQueue` should restart the task up to N times (with increased resources) if the task runs out of time or memory. Increased resources means longer time or more cores for the timed-out and out-of-memory cases, respectively.

The `MyQueue` *submit* sub-command makes it easy to submit thousands of tasks in a single command. As input *submit* takes a shell script, Python script, or Python module and executes the script/module in a number of folders. This makes it easy to submit a large number of tasks quickly. The *list* sub-command can then be used to monitor the execution of the tasks. Together with the *resubmit* sub-command it becomes easy to resubmit any tasks that might have failed. This example show how the sub-commands of `MyQueue` synergize and increase the efficiency of the user.

`MyQueue` has a simple Python interface that can be used to define workflows. A Python script defines a dependency tree of tasks that `MyQueue` can use to submit tasks without user involvement. The dependencies take the form "if task X is done then submit task Y". `MyQueue` works directly with folders and files, which makes it simple to use and easy to get started.

Compared to the current state of the field `MyQueue` distinguishes itself by focusing *not* on automatic handling of crashes but *only* on the single problem of submitting and managing thousands of tasks. In the scientific field of the authors, atomic-scale simulations, commonly used workflow managers are AiiDA (Pizzi, Cepellotti, Sabatini, Marzari, & Kozinsky, 2016) and Fireworks (Jain et al., 2015). Fireworks' centralized server model is advantageous when coordinating tasks distributed between multiple users. In constrast to Fireworks, `MyQueue` is

installed per user, is completely decentralized, and cannot coordinate tasks between multiple users. AiiDA is a fully automatic workflow tool designed to ensure data provenance. In contrast to AiiDA, `MyQueue` does not handle data provenance and does not focus on the automization of the workflow process. These design decisions can be seen both as a drawback and an advantage depending on the use case, but in any case makes `MyQueue` easier to learn. To summarize, `MyQueue` is a personal, decentralized, and lightweight front-end for schedulers with support for submitting workflows. It requires no system administrator and no database server.

`MyQueue` is useful for high-throughput computations, which require automatic submission of thousands of interdependent jobs. For example, `MyQueue` has been used to drive high-throughput screening studies coordinating on the order of 100,000 individual tasks (Haastrup et al., 2018), (Bölle et al., 2019). `MyQueue` is also used by the Atomic Simulation Recipes project, which is a library of tasks for atomic simulations.

Acknowledgments

K. S. T. acknowledges funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 773122, LIMA).

References

Bölle, F. T., Mathiesen, N. R., Nielsen, A. J., Vegge, T., Lastra, J. M. G., & Castelli, I. E. (2019). Autonomous Discovery of Materials for Intercalation Electrodes. doi:10.26434/chemrxiv.9971054.v1

Haastrup, S., Strange, M., Pandey, M., Deilmann, T., Schmidt, P. S., Hinsche, N. F., Gjerding, M. N., et al. (2018). The computational 2D materials database: High-throughput modeling and discovery of atomically thin crystals. *2D Materials*, *5*(4), 042002. doi:10.1088/2053-1583/aacfc1

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., et al. (2015). FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, *27*(17), 5037–5059. doi:10.1002/cpe.3505

Jette, M. A., Yoo, A. B., & Grondona, M. (2002). SLURM: Simple linux utility for resource management. In *In lecture notes in computer science: Proceedings of job scheduling strategies for parallel processing (JSSPP) 2003* (pp. 44–60). Springer-Verlag. doi:10.1007/10968987_3

Pizzi, G., Cepellotti, A., Sabatini, R., Marzari, N., & Kozinsky, B. (2016). AiiDA: Automated interactive infrastructure and database for computational science. *Computational Materials Science*, *111*, 218–230. doi:https://doi.org/10.1016/j.commatsci.2015.09.013