The Journal of Open Source Software

pysqa - Python Simple Queuing System Adapter

Jan Janssen [1] **and Joerg Neugebauer** [1]

1 Max Planck Institute for Sustainable Materials, Düsseldorf, Germany

DOI: 10.21105/joss.10961

Software

- Review
- Repository
- Archive

Editor: Anjali Sandip

Reviewers:

- @GuilloteauQ
- @jmukher1
- @CompEng0001

Submitted: 16 June 2026
Published: 10 September 2026

Summary

High-performance computing (HPC) resources are commonly accessed through batch scheduling systems such as LSF (*IBM Spectrum LSF Documentation*, 1992), Torque/PBS (*OpenPBS*, 1998), SGE (Gentzsch, 2001), SLURM (Jette et al., 2002), Moab (*Adaptive Computing*, 2003) and Flux (Ahn et al., 2014). These schedulers provide powerful command-line interfaces for job submission and resource management, but integrating scheduler interactions into Python applications often requires scheduler-specific scripts, shell wrappers, and custom parsing of scheduler outputs. As a result, scientific software frequently contains infrastructure code that is difficult to maintain and port across HPC environments.

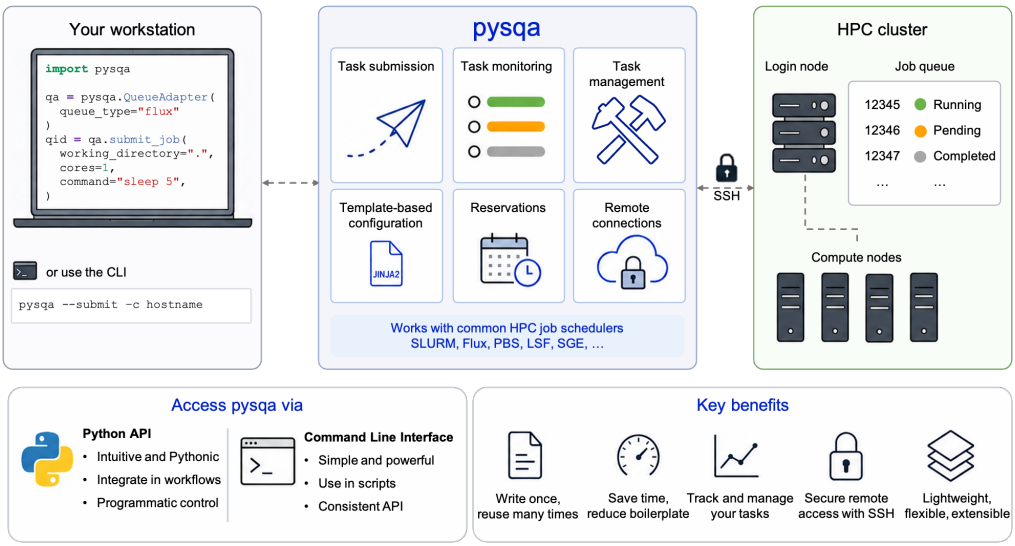

Figure 1: Schematic overview of pysqa. The package provides a scheduler-independent interface for submitting, monitoring, and managing HPC jobs from Python or the command line. Reusable Jinja2-based templates are translated to the native syntax of supported queuing systems (e.g., SLURM, PBS, LSF, and SGE), while built-in support for remote connections, reservations, and job management simplifies the integration of HPC resources into scientific workflows.

pysqa (Python Simple Queue Adapter) is an open-source Python library that provides a lightweight, scheduler-independent interface for submitting, monitoring, and managing HPC jobs. The design of pysqa is based on reusable queue templates written in the Jinja2 templating language (*Jinja*, 2008). Users define scheduler-specific submission scripts once and subsequently reuse them across many computational tasks and workflows. This approach separates scheduler configuration from application logic while preserving transparency by generating native scheduler submission scripts. pysqa currently supports the Flux, LSF, Moab, SGE, SLURM, and Torque queuing systems and includes optional support for remote job submission via SSH.

Unlike workflow orchestration frameworks, pysqa focuses exclusively on the scheduler interaction layer. It enables computational scientists and research software engineers to integrate HPC resources into Python applications with minimal complexity while retaining full control over scheduler-specific configurations. A schematic overview of pysqa is illustrated in Figure 1.

Statement of Need

Modern computational research increasingly relies on automated execution of simulations, machine learning workloads, and data-processing pipelines on shared HPC infrastructure. While scheduler command-line tools such as sbatch, squeue, and scancel provide direct access to HPC resources, scientific applications often require programmatic job submission and monitoring capabilities. Embedding scheduler-specific commands directly into software reduces portability and increases maintenance costs when users operate across multiple clusters. The primary audience for pysqa is developers of scientific software and research software engineers who require a lightweight, programmatic interface to HPC schedulers that can be used across different computing environments.

State of the field

Several software projects address related challenges. MyQueue (Mortensen et al., 2020) provides a higher-level task and workflow abstraction designed for scientific computing campaigns. PSI/J (Hategan-Marandiuc et al., 2023) offers a portable job execution API spanning multiple schedulers and execution backends. Jobflow-Remote focuses on remote execution of workflow graphs within the Jobflow ecosystem (Rosen et al., 2024). These tools provide broader workflow or interoperability capabilities, but they also introduce additional abstractions and infrastructure requirements.

pysqa addresses a different use case. It provides a minimal abstraction layer between Python applications and HPC schedulers while deliberately avoiding workflow management, databases, or orchestration services. The resulting design minimizes dependencies, simplifies deployment, and allows users to continue working with familiar scheduler submission scripts. This approach is particularly valuable for scientific software projects that require scheduler portability without adopting a complete workflow framework. Additionally, pysqa can also be implemented as a module in existing workflow frameworks (Janssen et al., 2019) and task schedulers (Janssen et al., 2025).

Software design

The central abstraction in pysqa is the QueueAdapter, which provides a scheduler-independent Python interface for submitting, monitoring, and managing jobs. Rather than invoking scheduler commands such as `sbatch`, `qsub`, `bsub`, or `flux submit` directly from shell scripts, users can interact with HPC resources through a small set of Python methods:

```
from pysqa import QueueAdapter

queue_adapter = QueueAdapter(directory="queues")

job_id = queue_adapter.submit_job(
    queue="standard",
    job_name="example",
    cores=16,
    run_time_max=3600,
    command="python simulation.py"
)
```

```
status = queue_adapter.get_queue_status()
queue_adapter.delete_job(job_id)
```

This Python interface simplifies the integration of HPC resources into scientific software, high-throughput simulation frameworks, and automated workflows. Resource requests and scheduler interactions can be expressed directly in Python without embedding scheduler-specific commands or parsing scheduler outputs. At the same time, pysqa does not replace the underlying scheduler command-line interface. Instead, it builds on top of existing scheduler functionality and generates standard submission scripts that remain fully transparent to users and administrators.

To achieve portability across different computing environments, pysqa separates scheduler configuration from submission script generation. Cluster-specific settings are defined in a YAML configuration file, which describes available queues, resource limits, and scheduler properties:

```yaml
queue_type: SLURM
queue_primary: standard

queues:
  standard:
    cores_min: 1
    cores_max: 128
    run_time_max: 86400
    script: slurm.sh
```

This configuration provides a machine-readable description of the HPC environment and can be shared across users and projects.

The scheduler submission scripts themselves are defined using Jinja2 templates. For example, a SLURM submission template can be written as:

```
#!/bin/bash
#SBATCH --job-name={{job_name}}
#SBATCH --ntasks={{cores}}
#SBATCH --time={{run_time_max}}
#SBATCH --output={{job_name}}.out

{{command}}
```

The use of Jinja2 templates preserves the familiar scheduler-native submission scripts that HPC users and administrators already maintain. Existing SLURM, PBS, or Flux scripts can typically be converted with only minor modifications by replacing fixed values with template variables. During job submission, pysqa combines the resource parameters provided through the Python interface with the cluster configuration and renders the corresponding scheduler script before submitting it through the scheduler's native command-line tools.

This separation of concerns provides three advantages. First, application developers interact with a consistent Python API independent of the underlying scheduler. Second, cluster-specific configuration is maintained centrally in YAML files rather than being embedded in application code. Third, scheduler experts retain full control over the generated submission scripts using familiar scheduler directives and scripting practices. As a result, pysqa combines the programmability of a Python interface with the transparency and flexibility of traditional scheduler-native workflows.

Research impact statement

pysqa was initially developed as a module of the pyiron workflow environment (Janssen et al., 2019). It was spun off into a standalone package to be used in different components of the pyiron ecosystem including executorlib (Janssen et al., 2025). Since the spin-off, external projects including ropt (Verveer, 2024), DREAMS (Wang et al., 2025), nipoppy (Bhagwat et al., 2026) and matsci-agent (Yaotang, 2026) have started to use pysqa.

Additional Details

The full documentation including a number of examples for the individual features is available at pysqa.readthedocs.io with the corresponding source code at github.com/pyiron/pysqa. pysqa is developed as an open-source library with a focus on stability.

AI usage disclosure

The initial version of pysqa was developed without the usage of AI. Github Co-pilot and Claude Code from Anthropic were used to extend type hints and docstrings and to enrich the documentation of pysqa. Finally, ChatGPT was used for writing the first draft of the manuscript, proofreading, and the conceptualization of the visuals.

Acknowledgements

J.J. and J.N. acknowledge funding from the Deutsche Forschungsgemeinschaft (DFG) through the CRC1394 "Structural and Chemical Atomic Complexity – From Defect Phase Diagrams to Material Properties", project ID 409476157.

References

Adaptive computing. (2003). https://adaptivecomputing.com/moab-hpc-suite

Ahn, D. H., Garlick, J., Grondona, M., Lipari, D., Springmeyer, B., & Schulz, M. (2014). Flux: A next-generation resource management framework for large HPC centers. *2014 43rd International Conference on Parallel Processing Workshops*, 9–17. https://doi.org/10.1109/ICPPW.2014.15

Bhagwat, N., Wang, M., Dugré, M., McPherson, B., & Poline, J.-B. (2026). *Nipoppy* (Version 0.4.6). Zenodo. https://doi.org/10.5281/zenodo.19379972

Gentzsch, W. (2001). Sun grid engine: Towards creating a compute power grid. *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 35–36. https://doi.org/10.1109/CCGRID.2001.923173

Hategan-Marandiuc, M., Merzky, A., Collier, N., Maheshwari, K., Ozik, J., Turilli, M., Wilke, A., Wozniak, J. M., Chard, K., Foster, I., Silva, R. F. da, Jha, S., & Laney, D. (2023). PSI/j: A portable interface for submitting, monitoring, and managing jobs. *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10. https://doi.org/10.1109/e-Science58273.2023.10254912

IBM spectrum LSF documentation. (1992). https://www.ibm.com/docs/en/spectrum-lsf/10.1.0

Janssen, J., Surendralal, S., Lysogorskiy, Y., Todorova, M., Hickel, T., Drautz, R., & Neugebauer, J. (2019). Pyiron: An integrated development environment for computational

materials science. *Computational Materials Science*, *163*, 24–36. https://doi.org/10.1016/j.commatsci.2018.07.043

Janssen, J., Taylor, M. G., Yang, P., Neugebauer, J., & Perez, D. (2025). Executorlib - up-scaling Python workflows for hierarchical heterogenous high-performance computing. *Journal of Open Source Software*, *10*(108), 7782. https://doi.org/10.21105/joss.07782

Jette, M. A., Yoo, A. B., & Grondona, M. (2002). SLURM: Simple Linux utility for resource management. *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, 44–60. https://doi.org/10.1007/10968987_3

Jinja. (2008). https://jinja.palletsprojects.com

Mortensen, J. J., Gjerding, M., & Thygesen, K. S. (2020). MyQueue: Task and workflow scheduling system. *Journal of Open Source Software*, *5*(45), 1844. https://doi.org/10.21105/joss.01844

OpenPBS. (1998). https://www.openpbs.org

Rosen, A. S., Gallant, M., George, J., Riebesell, J., Sahasrabuddhe, H., Shen, J.-X., Wen, M., Evans, M. L., Petretto, G., Waroquiers, D., Rignanese, G.-M., Persson, K. A., Jain, A., & Ganose, A. M. (2024). Jobflow: Computational workflows made simple. *Journal of Open Source Software*, *9*(93), 5995. https://doi.org/10.21105/joss.05995

Verveer, P. (2024). Ropt - a Python module for robust optimization. In *GitHub repository*. GitHub. https://github.com/tno-ropt/ropt

Wang, Z., Huang, H., Zhao, H., Xu, C., Zhu, S., Janssen, J., & Viswanathan, V. (2025). *DREAMS: Density functional theory based research engine for agentic materials simulation*. https://arxiv.org/abs/2507.14267

Yaotang, Z. (2026). Matsci-agent - a Hermes plugin for reusable materials-science tools. In *GitHub repository*. GitHub. https://github.com/chouyoudou/matsci-agent