kiwiPy: Robust, high-volume, messaging for big-data and computational science workflows

Uhrin, Martin; Huber, Sebastiaan P.

Published in:
The Journal of Open Source Software

Link to article, DOI:
10.21105/joss.02351

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Uhrin, M., & Huber, S. P. (2020). kiwiPy: Robust, high-volume, messaging for big-data and computational science workflows. The Journal of Open Source Software, 5(52), [2351]. https://doi.org/10.21105/joss.02351

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
kiwiPy: Robust, high-volume, messaging for big-data and computational science workflows

Martin Uhrin¹, 2, 3 and Sebastiaan P. Huber², 3

1 Department of Energy Conversion and Storage, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark 2 National Centre for Computational Design and Discovery of Novel Materials (MARVEL), École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland 3 Theory and Simulation of Materials (THEOS), Faculté des Sciences et Techniques de l’Ingénieur, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

Summary

The computational sciences have seen a huge increase in the use of high-throughput, automated, workflows over the course of the last two decades or so. Focusing on just our domain of computational materials science, there have been several large scale initiatives to provide high-quality results from standardised calculations (Curtarolo et al., 2012; Draxl & Scheffler, 2019; Jain et al., 2013; Landis et al., 2012; Saal, Kirklin, Aykol, Meredig, & Wolverton, 2013; Talirz et al., 2020). Almost all of these repositories are populated using results from high-throughput quantum mechanical calculations that rely on workflow frameworks (Jain et al., 2015; Mortensen, Gjerding, & Thygesen, 2020), including our own AiiDA (Huber et al., 2020; Pizzi, Cepellotti, Sabatini, Marzari, & Kozinsky, 2016) which powers the Materials Cloud. One of the many challenges for such frameworks is maximising fault-tolerance whilst simultaneously maintaining high-throughput, often across several systems (typically the client launching the tasks, the supercomputer carrying out the computations, and the server hosting the database).

On the software level, these problems are perhaps best addressed by using messaging brokers that take responsibility for guaranteeing the durability (or persistence) and atomicity of messages and often enable event-based communication. Indeed, solutions such as RabbitMQ, see widespread adoption in industry. However, adoption in academia has been more limited, with home-made queue data structures, race condition susceptible locks and polling based solutions being commonplace. This is likely due to message brokers typically having complex APIs (which reflect the non-trivial nature of the underlying protocol) as well as the lack of familiarity with event-based programming in general within the community. KiwiPy was designed specifically to address both these issues, by providing a tool that enables building robust, event-based systems with an interface that is as simple as possible.

In kiwiPy, all messages are saved to disk by RabbitMQ, meaning that any or all systems involved in a workflow, including the broker, can be shut down (abruptly or gracefully), and the previous state can be recreated allowing the workflow to continue when the necessary resources are brought back online. This is especially important for long-running HPC jobs that may take days or weeks. This feature differentiates kiwiPy from protocols such as MPI, ZeroMQ or libraries such as Dask, which do not persist their state.

A number of libraries for interacting directly with message brokers exist, including Pika, aio-pika, py-amqp, kombu and others. These tend to be rather low-level and focus on channels, exchanges, routing, sockets and so on. A comparison of the difference in focus between Pika and kiwiPy can be found in the documentation. At the other end of the spectrum are libraries such as Celery, RQ and others that provide task queues and libraries such as RPyC,
Spyne, Python-JRPC and others that enable remote procedure calls. In contrast, kiwiPy brings together three commonly used message types (task queues, Remote Procedure Calls (RPCs), and, broadcasts) in a single interface.

All messaging in kiwiPy is done in the Communicator class, which can be trivially constructed by providing a URI string pointing to the RabbitMQ server. By default, kiwiPy creates a separate communication thread that the user never sees, allowing them to interact with the communicator using familiar Python syntax, without the need to be familiar with either coroutines or multithreading. This has the additional advantage that kiwiPy will maintain heartbeats (a periodic check to make sure the connection is still alive) with the server whilst the user code can be doing other things. Heartbeats are an essential part of RabbitMQ’s fault tolerance; two missed checks will automatically trigger the message to be requeued to be picked up by another client.

To demonstrate some of the possible usage scenarios, we briefly outline the way kiwiPy is used in AiiDA. AiiDA, amongst other things, manages the execution of complex workflows made up of processes that may have checkpoints.

Task queues

As is common for high-throughput workflow engines, AiiDA maintains a task queue to which processes are submitted (typically from the user’s workstation). These tasks are then consumed by multiple daemon processes (which may also be on the user’s workstation or remote) and will only be removed from the task queue once they have been acknowledged to be completed by the consumer. The daemon can be gracefully or abruptly shut down and no task will be lost, since the task will simply be requeued by the broker once it notices that the consumer has died. Furthermore, there are no worries about race conditions between multiple daemon processes, since the task queue is guaranteed to only distribute each task to, at most, one consumer at a time.

Remote Procedure Calls

These are used to control live processes. Each process has a unique identifier and can be sent a pause, play, or kill message, the response to which is optionally sent back to the initiator to indicate success or something else.

Broadcasts

These currently serve two purposes: sending pause, play, or kill messages to all processes at once by broadcasting the relevant message, and controlling the flow between processes. If a parent process is waiting for a child to complete, it will be informed of this via a broadcast message from the child saying that its execution has terminated. This enables decoupling, as the child need not know about the existence of the parent.

Together these three message types allow AiiDA to implement a highly-decoupled, distributed, yet, reactive system that has proven to be scalable from individual laptops to workstations, driving simulations on high-performance supercomputers with workflows consisting of varying durations, ranging from milliseconds up to multiple days or weeks.

It is our hope that by lowering the barriers to adoption, kiwiPy will bring the benefits of industry grade message brokers to academia and beyond, ultimately making robust scientific software easier to write and maintain.

Uhrin et al., (2020). kiwiPy: Robust, high-volume, messaging for big-data and computational science workflows. Journal of Open Source Software, 5(52), 2351. https://doi.org/10.21105/joss.02351
Acknowledgements

We would like to thank Giovanni Pizzi, Nicola Marzari, and the AiiDA team for their continuous coordination and development of the project. We also thank Jason Yu for contributing the first version of the documentation. This work is supported by the MARVEL National Centre for Competency in Research funded by the Swiss National Science Foundation (grant agreement ID 51NF40-182892) and the European Materials Modelling Council-CSA funded by the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No 723867.

References

Curtarolo, S., Setyawan, W., Hart, G. L. W., Jahnatek, M., Chepulskii, R. V., Taylor, R. H., Wang, S., et al. (2012). AFLOW: An automatic framework for high-throughput materials discovery. *Computational Materials Science, 58*, 218–226. doi:10.1016/j.commatsci.2012.02.005

Draxl, C., & Scheffler, M. (2019). The NOMAD laboratory: from data sharing to artificial intelligence. *Journal of Physics: Materials, 2*(3), 036001. doi:10.1088/2515-7639/ab13bb

Huber, S. P., Zoupanos, S., Uhrin, M., Talirz, L., Kahle, L., Häuselmann, R., Gresch, D., et al. (2020). AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance. Retrieved from http://arxiv.org/abs/2003.12476

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

Jain, A., Ong, S. P., Hautier, G., Chen, W., Richards, W. D., Dacek, S., Cholia, S., et al. (2013). The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials, 1*(1), 011002. doi:10.1063/1.4812323

Landis, D. D., Hummelshøj, J. S., Nestorov, S., Greeley, J., Dulak, M., Bligaard, T., Norskov, J. K., et al. (2012). The Computational Materials Repository. *Computing in Science & Engineering, 14*(6), 51–57. doi:10.1109/MCSE.2012.16

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

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:10.1016/j.commatsci.2015.09.013

Saal, J. E., Kirklin, S., Aykol, M., Meredig, B., & Wolverton, C. (2013). Materials Design and Discovery with High-Throughput Density Functional Theory: The Open Quantum Materials Database (OQMD). *JOM, 65*(11), 1501–1509. doi:10.1007/s11837-013-0755-4

Talirz, L., Kumbhar, S., Passaro, E., Yakutovich, A. V., Granata, V., Gargiulo, F., Borelli, M., et al. (2020). Materials Cloud, a platform for open computational science, 1–22. Retrieved from http://arxiv.org/abs/2003.12510

Uhrin et al., (2020). kiwiPy: Robust, high-volume, messaging for big-data and computational science workflows. *Journal of Open Source Software, 5*(52), 2351. https://doi.org/10.21105/joss.02351