funsies: A minimalist, distributed and dynamic workflow engine

Cyrille Lavigne[*1] **and Alán Aspuru-Guzik**[1, 2, 3, 4]

1 Department of Computer Science, University of Toronto, 40 St. George St, Toronto, Ontario M5S 2E4, Canada **2** Chemical Physics Theory Group, Department of Chemistry, University of Toronto, 80 St. George St, Toronto, Ontario M5S 3H6, Canada **3** Vector Institute for Artificial Intelligence, 661 University Ave Suite 710, Toronto, Ontario M5G 1M1, Canada **4** Lebovic Fellow, Canadian Institute for Advanced Research (CIFAR), 661 University Ave, Toronto, Ontario M5G, Canada

DOI: 10.21105/joss.03274

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @gflofst
- @vijaysm

Submitted: 28 April 2021
Published: 02 October 2021

Summary

Large-scale, high-throughput computational investigations are increasingly common in chemistry and physics. Until recently, computational chemistry was primarily performed using all-in-one monolithic software packages (Aprà et al., 2020; Aquilante et al., 2020; Barca et al., 2020; Kühne et al., 2020; Romero et al., 2020; Smith et al., 2020). However, the limits of individual programs become evident when tackling complex multifaceted problems. As such, it is increasingly common to use multiple disparate software packages in a single computational pipeline, often stitched together using shell scripts in languages such as Bash, or using Python and other interpreted languages.

These complex computational pipelines are difficult to scale and automate, as they often include manual steps and significant "human-in-the-loop" tuning. Shell scripting errors are often undetected, which can compromise scientific results. Conversely, exception-based error handling, the standard approach in Python, can readily bring a computational workflow to a halt when exceptions are not properly caught (Weimer & Necula, 2008).

`funsies` is a set of Python programs and modules to describe, execute and analyze computational workflows, with first-class support for shell scripting. It includes a lightweight, decentralized workflow engine backed by a NoSQL store. Using `funsies`, external programs and Python-based computations are easily mixed together. Errors are detected and propagated throughout computations. Automatic, transparent incremental computing (based on a hash tree data structure) provides a convenient environment for iterative prototyping of computationally expensive workflows.

Statement of need

Modern workflow management programs used in the private sector, such as Apache Airflow and Uber's Cadence, are robust and extremely scalable, but are difficult to deploy. Scientific workflow management systems, many of which are compiled in (Tommaso, 2021) and systematically reviewed in (Mölder et al., 2021), are easier to set up on high-performance computing clusters, but are tuned to the needs of specific disciplines, such as bioinformatics or machine learning. This includes, for example, the use of configuration file formats (YAML, JSON, etc.), packaging tools (for example, conda or Docker), locked-in compute providers (Amazon Web Services, Google Cloud) and storage formats that may be common in specific scientific fields but not throughout the greater community.

[*] Corresponding author.

For our own group's research program, we wanted to have available a lightweight workflow management system that could be readily deployed to new and varied computational facilities and local workstations with minimal effort. This system had to support our existing shell-based and Python-based scripts, and be flexible enough for rapid prototyping all the way to large-scale computational campaigns, and provide an embeddable solution that can be bundled within other software (Lavigne et al., 2020). Finally, we were looking for a tool that could integrate data generation and storage, to avoid the common practice of transforming the filesystem into what is effectively a schema-less database. We developed `funsies` to address those needs.

Features and Implementation

`funsies` is a Python library and a set of associated command-line tools. Using the `funsies` library, general computational workflows are described in lazily evaluated Python code. Operations in `funsies` are taken to be pure, that is, all operation outputs are entirely and solely determined by their inputs. Workflows are orchestrated using Python by manipulating pointers to yet-to-be-calculated data. Workflow instructions are transparently translated and saved as graph elements in a Redis database.

Computational evaluation is initiated by the user asking for specific output value. The task graph from these final outputs is walked back all the way to those operations with no dependencies. These initial operations are then queued for execution. Lightweight worker processes, instantiated from the command line on local or remote machines, connect to the Redis database and start executing the workflow. For each operation, the worker checks if outputs are already cached, and if not, executes the associated function and saves its outputs. It then enqueues any dependents for execution, by itself or by other workers. In this way, the entire computational graph is evaluated in a distributed, decentralized fashion without any scheduler or manager program. Errors in workflows are handled using a functional approach inspired by Rust (Klabnik & Nichols, 2019). Specifically, exceptions are propagated through workflow steps, canceling dependent tasks, without interrupting valid workflow branches. This provides both easy error tracing and a high degree of fault tolerance.

The main distinguishing feature of `funsies` is the hash tree structure that is used to encode all operations and their inputs. The causal hashing approach used in `funsies` can also be found in Snakemake (Mölder et al., 2021) as an optional component and the (now defunct) Koji workflow system (Maymounkov, 2018), as part of the Nix package manager (Dolstra et al., 2004) and in the Git version control system (Chacon & Straub, 2014). In `funsies`, we replace all filesystem operations with hash addressed operations; that is all I/O operations and dependencies are tracked.

Every operation has a hash address that is computed from the hash values of its dependencies and a hashed identifier for the associated operation on data. In this way, the consistency of data dependencies is strongly enforced. Changes to data and operations are automatically and transparently propagated, as changing a single dependency will cause a rehash of all its dependents, effectively producing a new workflow with no associated data that needs to be recomputed. Alternatively, if data already exists at a specific hash address, then it was generated from the same operations that produced that hash. In this way, the hash tree structure enables transparent and automatic incremental recomputing.

Using hash addresses also enables decentralization, as we can rely on the unlikeliness of hash collisions (Stevens et al., 2017) to eliminate centralized locks. An important advantage of this approach is that it allows worker processes to generate their own workflows of tasks dynamically. Results from these dynamic workflows can be collected and used further in the workflow description, provided they can be reduced to a number of outputs known at compile time, a technique similar to MapReduce (Dean & Ghemawat, 2004).

As of now, we have published one project (Pollice et al., 2021) that used an earlier iteration of `funsies`, and are using it in multiple ongoing inquiries. We provide several sample workflows on GitHub, with a focus on computational chemistry, quantum computing, and high-performance computing infrastructure.

We intend to maintain `funsies` and of course welcome collaborations from contributors around the world.

Acknowledgements

We acknowledge testing by early users Cher-Tian Ser, Kjell Jorner and Gabriel dos Passos Gomes. CL also thanks Chris Crebolder for help setting up documentation pages. We acknowledge the Defense Advanced Research Projects Agency (DARPA) under the Accelerated Molecular Discovery Program under Cooperative Agreement No. HR00111920027 dated August 1, 2019. The content of the information presented in this work does not necessarily reflect the position or the policy of the Government. A. A.-G. thanks Dr. Anders G. Frøseth for his generous support. A. A.-G. also acknowledges the generous support of Natural Resources Canada and the Canada 150 Research Chairs program. We thank Compute Canada for providing computational resources.

References

Aprà, E., Bylaska, E. J., Jong, W. A. de, Govind, N., Kowalski, K., Straatsma, T. P., Valiev, M., Dam, H. J. J. van, Alexeev, Y., Anchell, J., Anisimov, V., Aquino, F. W., Atta-Fynn, R., Autschbach, J., Bauman, N. P., Becca, J. C., Bernholdt, D. E., Bhaskaran-Nair, K., Bogatko, S., … Harrison, R. J. (2020). NWChem: Past, present, and future. *Journal of Chemical Physics*, *152*(18), 184102. https://doi.org/10.1063/5.0004997

Aquilante, F., Autschbach, J., Baiardi, A., Battaglia, S., Borin, V. A., Chibotaru, L. F., Conti, I., De Vico, L., Delcey, M., Fdez. Galván, I., Ferré, N., Freitag, L., Garavelli, M., Gong, X., Knecht, S., Larsson, E. D., Lindh, R., Lundberg, M., Malmqvist, P. Å., … Veryazov, V. (2020). Modern quantum chemistry with [Open]Molcas. *Journal of Chemical Physics*, *152*(21), 214117. https://doi.org/10.1063/5.0004835

Barca, G. M. J., Bertoni, C., Carrington, L., Datta, D., De Silva, N., Deustua, J. E., Fedorov, D. G., Gour, J. R., Gunina, A. O., Guidez, E., Harville, T., Irle, S., Ivanic, J., Kowalski, K., Leang, S. S., Li, H., Li, W., Lutz, J. J., Magoulas, I., … Gordon, M. S. (2020). Recent developments in the general atomic and molecular electronic structure system. *Journal of Chemical Physics*, *152*(15), 154102. https://doi.org/10.1063/5.0005188

Chacon, S., & Straub, B. (2014). *Pro git*. Apress. ISBN: 978-1-4842-0076-6

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 137–150.

Dolstra, E., Jonge, M. de, & Visser, E. (2004). *Nix: A safe and policy-free system for software deployment*. 14. https://edolstra.github.io/pubs/nspfssd-lisa2004-final.pdf

Klabnik, S., & Nichols, C. (2019). *The Rust programming language (covers Rust 2018)*. No Starch Press. ISBN: 978-1-71850-044-0

Kühne, T. D., Iannuzzi, M., Del Ben, M., Rybkin, V. V., Seewald, P., Stein, F., Laino, T., Khaliullin, R. Z., Schütt, O., Schiffmann, F., Golze, D., Wilhelm, J., Chulkov, S., Bani-Hashemian, M. H., Weber, V., Borštnik, U., Taillefumier, M., Jakobovits, A. S., Lazzaro, A., … Hutter, J. (2020). CP2K: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations. *Journal of Chemical Physics*, *152*(19), 194103. https://doi.org/10.1063/5.0007045

Lavigne, C., Passos Gomes, G. dos, Pollice, R., & Aspuru-Guzik, A. (2020). *Automatic discovery of chemical reactions using imposed activation*. https://doi.org/10.26434/chemrxiv.13008500.v2

Maymounkov, P. (2018). Koji: Automating pipelines with mixed-semantics data sources. *arXiv:1901.01908 [cs]*. http://arxiv.org/abs/1901.01908

Mölder, F., Jablonski, K., Letcher, B., Hall, M., Tomkins-Tinch, C., Sochat, V., Forster, J., Lee, S., Twardziok, S., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., & K ster, J. (2021). Sustainable data analysis with snakemake [version 2; peer review: 1 approved, 1 approved with reservations]. *F1000Research*, *10*(33). https://doi.org/10.12688/f1000research.29032.2

Pollice, R., Friederich, P., Lavigne, C., Gomes, G. dos P., & Aspuru-Guzik, A. (2021). Organic molecules with inverted gaps between first excited singlet and triplet states and appreciable fluorescence rates. *Matter*. https://doi.org/10.1016/j.matt.2021.02.017

Romero, A. H., Allan, D. C., Amadon, B., Antonius, G., Applencourt, T., Baguet, L., Bieder, J., Bottin, F., Bouchet, J., Bousquet, E., Bruneval, F., Brunin, G., Caliste, D., Côté, M., Denier, J., Dreyer, C., Ghosez, P., Giantomassi, M., Gillet, Y., … Gonze, X. (2020). ABINIT: Overview and focus on selected capabilities. *Journal of Chemical Physics*, *152*(12), 124102. https://doi.org/10.1063/1.5144261

Smith, D. G. A., Burns, L. A., Simmonett, A. C., Parrish, R. M., Schieber, M. C., Galvelis, R., Kraus, P., Kruse, H., Di Remigio, R., Alenaizan, A., James, A. M., Lehtola, S., Misiewicz, J. P., Scheurer, M., Shaw, R. A., Schriber, J. B., Xie, Y., Glick, Z. L., Sirianni, D. A., … Sherrill, C. D. (2020). PSI4 1.4: Open-source software for high-throughput quantum chemistry. *Journal of Chemical Physics*, *152*(18), 184108. https://doi.org/10.1063/5.0006002

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017). The first collision for full SHA-1. In J. Katz & H. Shacham (Eds.), *Advances in cryptology – CRYPTO 2017* (pp. 570–596). Springer International Publishing. https://doi.org/10.1007/978-3-319-63688-7_19

Tommaso, P. D. (2021). A curated list of awesome pipeline toolkits inspired by awesome sysadmin. In *GitHub repository*. GitHub. https://github.com/pditommaso/awesome-pipeline

Weimer, W., & Necula, G. C. (2008). Exceptional situations and program reliability. *ACM Transactions on Programming Languages and Systems*, *30*(2), 8:1–8:51. https://doi.org/10.1145/1330017.1330019