The Journal of Open Source Software

Melissa: coordinating large-scale ensemble runs for deep learning and sensitivity analyses

Marc Schouler [1], **Robert Alexander Caulk** [1], **Lucas Meyer** [1,2], **Théophile Terraz**[1], **Christoph Conrads**[1], **Sebastian Friedemann**[1], **Achal Agarwal** [1], **Juan Manuel Baldonado**[1], **Bartłomiej Pogodziński**[3], **Anna Sekuła** [3], **Alejandro Ribes**[2], **and Bruno Raffin**[1]

1 Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, France **2** Industrial AI Laboratory SINCLAIR, EDF Lab Paris-Saclay, France **3** Institute of Bioorganic Chemistry Polish Academy of Sciences, Poznań Supercomputing and Networking Center

DOI: 10.21105/joss.05291

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @acrlakshman
- @NoujoudNader

Submitted: 17 February 2023
Published: 16 June 2023

Statement of need

Large-scale ensemble runs typically consist of executing thousands of physical simulation instances according to a range of different input parameters. These ensemble runs enable sensitivity analyses, deep surrogate trainings, reinforcement learning, and data assimilation, but they rely on volumes of data that are too large to store. For example, a recent data assimilation ensemble study generated 1.3 PB of data (Yashiro et al., 2020). These enormous volumes of data hinder scientific analyses in two ways: first, the I/O speeds are the slowest component in supercomputers; the incongruence between slow read/write speeds compared to the rapid generation of data leads to a degradation and plateau of performance. Second, the file systems on supercomputers are not designed to allocate such large volumes of data to singular studies. To avoid this I/O limitation, scientists reduce their study size by running low resolution simulations or down-sampling output data in space and time. However, the I/O problem only becomes more pronounced as the speed and size of supercomputers continues to advance faster than I/O speeds of storage disks.

Summary

Melissa is a file avoiding, fault tolerant and elastic framework, generalized to perform ensemble runs such as *large scale sensitivity analysis* and *large scale deep surrogate training* on supercomputers. Some of the largest Melissa studies so far employed up to 30k cores to execute 80 000 parallel simulations while avoiding up to 288 TB of intermediate data storage (see (Ribés et al., 2022)). These large-scale studies avoid intermediate file storage due to Melissa's "online" (also referred to as in-transit and on-the-fly) data handling approach. As shown in (Fig. 1), Melissa's architecture relies on three interacting components, the launcher, the server, and the client:

1. Melissa client: the parallel numerical simulation code turned into a client. Each client sends its output to the server as soon as available. Clients are independent jobs.
2. Melissa server: a parallelized process in charge of processing the data upon arrival from the distributed and parallelized clients (*e.g.* computing statistics or training a neural network).
3. Melissa Launcher: the front-end Python script in charge of orchestrating the execution of the study. This piece of code interacts directly with `OpenMPI` or with the cluster scheduler (*e.g.* `slurm` or `OAR`) to submit and monitor the proper execution of all instances.

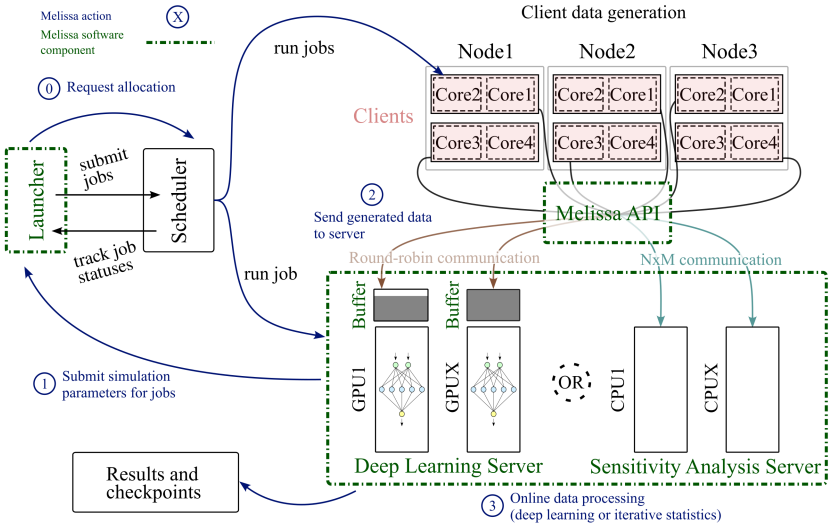

Figure 1: Melissa architecture. Specificities of sensitivity and deep learning applications appear side by side.

The Melissa server component is designed to be specialized for various types of ensemble runs:

Sensitivity Analysis (melissa-sa)

Melissa's sensitivity analysis server is built around two key concepts: iterative (sometimes also called incremental) statistics algorithms and asynchronous client/server model for data transfer. Simulation outputs are never stored on disk. Instead, they are sent via NxM communication patterns from the simulations to a parallelized server (Fig. 1). This method of data aggregation enables the calculation of rapid statistical fields in an iterative fashion, without storing any data to disk. Avoiding disk storage opens up the ability to compute oblivious statistical maps for all mesh elements, for every time step and on a full resolution study. Melissa comes with iterative algorithms for computing various statistical quantities (*e.g.* mean, variance, skewness, kurtosis and Sobol indices) and can easily be extended with new algorithms.

Deep Surrogate Training (melissa-dl)

Melissa's deep learning server adopts a similar philosophy. Clients communicate data in a round-robin fashion to the parallelized server (Fig. 1). The multi-threaded server then puts and pulls data samples in and out of a buffer (Fig. 2) which is used for building training batches. Melissa can perform data distributed parallelism training on several GPUs, associating a buffer to each of them. To ensure a proper memory management during execution, samples are selected and evicted according to a predefined policy. This strategy enables the online training method shown in Fig. 2. Furthermore, the Melissa architecture is designed to accommodate popular deep learning libraries such as PyTorch or Tensorflow.

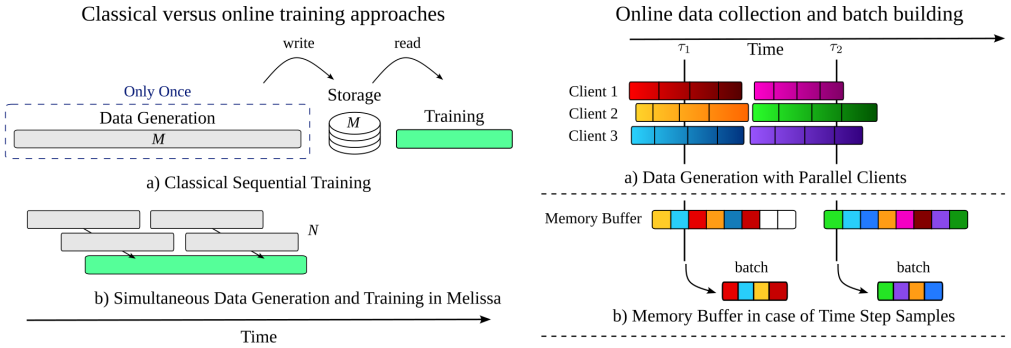

Figure 2: Overview of Melissa's deep learning framework (Meyer et al., 2023)

Use in academia

The original Sensitivity Analysis framework was published by Terraz et al. (2017), which set a foundation for a variety of subsequent studies. Ribés et al. (2019) used melissa for parameter augmented ensembles of curves, Ribés & Raffin (2020) used Melissa to help demonstrate challenges of in-situ analyses, and Friedemann & Raffin (2022) used the architecture of melissa to build a data assimilation software. Tangential work includes Guilloteau et al. (2022), which coupled melissa with NixOS to demonstrate distributed environments. More recently, Melissa's Deep Surrogate training was used to demonstrate improved training compared to offline analogs (Meyer et al., 2023).

State of the field

Melissa is unique in many ways, but there are a group of other open-sourced codes aiming to help scientists manage large scale analyses on supercomputers. For example, Merlin (*Merlin*, 2022) and Radical Pilot (Merzky et al., 2021) are supercomputing tools designed to help reduce friction in large scale ensemble runs dependent on file system I/O. Meanwhile, there exists a group of frameworks aimed at distributing python processes across clusters including Ray (Moritz et al., 2017) and Dask (Dask Development Team, 2016), but they do not support MPI based applications and are not file avoiding. Finally, there exist a group of in-situ processing tools which do not support ensemble runs including DataSpace (Docan et al., 2010), Decaf (Yildiz et al., 2022), and Damaris (Dorier et al., 2012). Although all these softwares are useful for particular applications, they do not fulfill all three main tasks Melissa was built for: large scale data generation, scheduler handling and file-avoiding data processing.

Using Melissa

Installing Melissa

Melissa includes an online documentation geared for new and advanced users alike. For example, installation instructions help users get started no matter which supercomputer they are working on. The typical installation is done via a `cmake` command. However, a `spack` install is also available.

Configuring Melissa

As highlighted in the documentation, running a Melissa analysis requires the user to:

1. Instrument the simulation code with the Melissa API (3 base calls: init, send and finalize) so it can become a Melissa client.

- Typically the calls to the `melissa_send()` are performed inside the simulation loop. For example, on each time step of a physical simulation may contain `melissa_send()` where it sends the physical quantities associated with domain at that time-step. This data will be the data that Melissa `server` collects and analyzes in an online fashion (iterative statistics or online training).
- As of now, Melissa provides an API compatible with solvers developed in the most popular HPC languages: C, Fortran and Python.

2. Configure the analysis. This includes defining the design of experiment (*i.e.* how to draw the parameters for each simulation execution), selecting which statistics to compute or specifying the Neural Network architecture, the training algorithm and parameters in case of deep-surrogate training.
   - The Melissa interface is comprised of two components, the configuration file (`config.json`) and the custom user class (`custom_server.py`). The configuration file is a json dictionary that contains all the study controls (*e.g.* number of clients to launch, which statistics to compute, batch_size etc.). `config.json` also contains instructions on how to execute the instrumented solver as well as all the custom launcher controls for the user's specific scheduler. Meanwhile, the `custom_server.py` is where a user customizes the machinery inside Melissa. For example, the `custom_server.py` may include specific deep-learning training loops/network architectures, custom iterative statistics, pre- and post-processing steps for the data, intermediate logging etc.
3. Start the Melissa launcher on the terminal or on the front-end of the supercomputer. Melissa takes care of requesting resources to execute the server and runner, monitoring the execution, and restarting failing components when necessary.

Running Melissa

After the user has instrumented their simulation code and configured their custom server, the study is launched with a single command:

```
melissa-launcher -c config.json
```

Monitoring Melissa

Melissa also contains a variety of monitoring/logging features to help users track live studies and post-processes completed studies. One feature is called the `melissa monitor` which is designed to run in terminals directly on supercomputers. This feature displays the number of waiting, running, terminated, and failed jobs. Meanwhile, for deep-learning studies, Melissa has tensorboard integration which allows users to track the training loss and other custom metrics in real-time.

Melissa test suite and CI

The Melissa source code contains a robust CI which builds the source, builds/publishes the documentation, runs unit tests, and runs full integration tests. This CI serves to maintain code quality while advancing developments in an open-sourced fashion between a group of developers.

Examples and exhibits

Melissa was already successfully coupled with state-of-the-art PDE solvers (*e.g.* Code-Saturne, FEniCS) and the source code provides ready to use examples of the heat equation and the Lorenz system. These examples include training deep-learning surrogates using distributed GPUs, and iterative statistics. Further, Melissa includes a fully reproducible online vs offline

deep learning comparison. Finally, if users seek active support they are encouraged to join our Discourse forum and ask questions to the development team.

Acknowledgment

The development of Melissa was made possible thanks to several funding sources that include:

- EOCOE II: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824158
- REGALE: This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956560

References

Dask Development Team. (2016). *Dask: Library for dynamic task scheduling*. https://dask.org

Docan, C., Parashar, M., & Klasky, S. (2010). Dataspaces: An interaction and coordination framework for coupled simulation workflows. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 25–36. https://doi.org/10.1145/1851476.1851481

Dorier, M., Antoniu, G., Cappello, F., Snir, M., & Orf, L. (2012, September). Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. *CLUSTER 2012 - IEEE International Conference on Cluster Computing*. https://doi.org/10.1109/CLUSTER.2012.26

Friedemann, S., & Raffin, B. (2022). An elastic framework for ensemble-based large-scale data assimilation. *The International Journal of High Performance Computing Applications*, *36*(4), 543–563. https://doi.org/10.1177/10943420221110507

Guilloteau, Q., Bleuzen, J., Poquet, M., & Richard, O. (2022). Painless transposition of reproducible distributed environments with NixOS compose. *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 1–12. https://doi.org/10.1109/CLUSTER51413.2022.00051

Merlin. (2022). https://merlin.readthedocs.io/en/latest/index.html

Merzky, A., Turilli, M., Titov, M., Al-Saadi, A., & Jha, S. (2021). Design and performance characterization of RADICAL-pilot on leadership-class platforms. *CoRR*, *abs/2103.00091*. https://doi.org/10.1109/TPDS.2021.3105994

Meyer, L., Schouler, M., Caulk, R. A., Ribes, A., & Raffin, B. (2023). Training deep surrogate models with large scale online learning. *2023 ICML International Conference of Machine Learning*, Accepted.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Paul, W., Jordan, M. I., & Stoica, I. (2017). Ray: A distributed framework for emerging AI applications. *CoRR*, *abs/1712.05889*. http://arxiv.org/abs/1712.05889

Ribés, A., Pouderoux, J., & Iooss, B. (2019). A visual sensitivity analysis for parameter-augmented ensembles of curves. *Journal of Verification, Validation and Uncertainty Quantification*, *4*(4). https://doi.org/10.1115/1.4046020

Ribés, A., & Raffin, B. (2020). The Challenges of In Situ Analysis for Multiple Simulations. *ISAV 2020 - In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 1–6. https://doi.org/10.1145/3426462.3426468

Ribés, A., Terraz, T., Fournier, Y., Iooss, B., & Raffin, B. (2022). Unlocking large scale uncertainty quantification with in transit iterative statistics. In H. Childs, J. C. Bennett, &

C. Garth (Eds.), *In situ visualization for computational science* (pp. 113–136). Springer International Publishing. https://doi.org/10.1007/978-3-030-81627-8_6

Terraz, T., Ribes, A., Fournier, Y., Iooss, B., & Raffin, B. (2017). Melissa: Large Scale In Transit Sensitivity Analysis Avoiding Intermediate Files. *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC17)*, 1–14. https://hal.inria.fr/hal-01607479

Yashiro, H., Terasaki, K., Kawai, Y., Kudo, S., Miyoshi, T., Imamura, T., Minami, K., Inoue, H., Nishiki, T., Saji, T., Satoh, M., & Tomita, H. (2020). A 1024-member ensemble data assimilation with 3.5-km mesh global weather simulations. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–10. https://doi.org/10.1109/SC41405.2020.00005

Yildiz, O., Dreher, M., & Peterka, T. (2022). Decaf: Decoupled dataflows for in situ workflows. In H. Childs, J. C. Bennett, & C. Garth (Eds.), *In situ visualization for computational science* (pp. 137–158). Springer International Publishing. ISBN: 978-3-030-81627-8