Journal of Physics:
Conference Series

PURPOSE-LED PUBLISHING™

PAPER • OPEN ACCESS

A framework for data processing used at the Hard X-ray Micro/Nano Probe at PETRA III based on micro-services.

To cite this article: J Garrevoet 2025 *J. Phys.: Conf. Ser.* **3010** 012136

View the article online for updates and enhancements.

You may also like

- Impacts on compound drought heatwave events in Australia per global warming level
Sarah Chapman, Ralph Trancoso, Jozef Syktus et al.
- A MUSIC-based Method for Detection and Localization of UAV Swarm
Ninghui Li, Xiaokuan Zhang, Weike Feng et al.
- Urban Expansion Patterns and Metropolitan Development: A Comparative Study of Malang Raya and Surabaya Metropolitan Area
F Firmansyah, M H Farras, R Z F Sihotang et al.

This content was downloaded from IP address 131.169.254.62 on 11/06/2025 at 13:57

Journal of Physics: Conference Series **3010** (2025) 012136 doi:10.1088/1742-6596/3010/1/012136

A framework for data processing used at the Hard X-ray Micro/Nano Probe at PETRA III based on micro-services.

J Garrevoet[1]

[1]Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

E-mail: jan.garrevoet@desy.de

Abstract. These days, experiments are more complex, instruments more complicated, requirements and configurations are always changing, and beamline users are less familiar with the technical and methodological side of the beamline. So, it is important to process the data for the users so they can focus on interpreting the results. Having a system in place to quickly provide feedback for the users is very useful and helps to make efficient use of a beamline.

1 Motivation

Instruments are becoming more and more specialised to allow for more complex techniques. The complexity is often increased by the wish of the scientists to not solely focus on a single methodology but perform multi-modal experiments. Although several beamlines around the world offer similar capabilities, the technical implementation will each time be different. These differences in implementation are pushed further down to the software that is responsible of the data analysis. Generally speaking, every single datasets needs to be pre- and post processed in a specific way before it can be interpreted by the visiting scientists (users). Therefore, shifting the responsibility to the beamline scientists or supporting scientists makes perfect sense. They are experts in the field and when changes are made, whether on hardware or software side, only one software stack needs to be adapted. In the past this would have caused every user piece of code to break, resulting in extra time investment on both the beamline scientist side as well as on the user side. The large intervals between user visits, different coding languages, and code quality, complicated the efforts to keep user code compatible with the beamline.

2 The approach

The approach to tackle this problem will always stir up some debate on what is the correct way to solve this problem. It would be naive to say that there is a single solution to this problem, which is supported by the large number of work flow systems that are available, like Apache Airflow (Apache (2024)), Orange Canvas (OrangeDataMining (2024)), and Dask (Dask (2024)). None of the available workflow systems does offer every feature.

One typically categorises these systems depending on their deployment.

The first strategy is to start a single pipeline that takes care of every single individual processing step that would convert raw data into analysed data. The other way to deploy a system like this would follow the schema of splitting every single processing step into a separate entity, often referred to as micro-services.

Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

Journal of Physics: Conference Series **3010** (2025) 012136 doi:10.1088/1742-6596/3010/1/012136

Besides the obvious architectural differences, operationally they also differ greatly by the fact that the former will each time be started when there is a job that needs to be executed while the latter will be persistently running and can be shared among different workflows.

The framework presented here is based on the latter. Since every function runs as a micro-service it offers a lot of flexibility regarding scaling and deployment, where the down side is complexity. The main language used is Python, but lower level languages are used when performance is key.

3 The framework

3.1 The core components

The entire framework is event driven and components are connected via ZMQ (ZeroMQ (2024)). Every component is configured via a yaml file that stipulates which modules to load and what the upstream and downstream components it should expect.

The top component in the stack is the topology manager. As the name suggests, it tracks the name and configuration of every single component that is part of the entire system. It is the responsibility of every component to register itself with the topology manager and make its configuration available. Moreover, it provides a convenient way to propagate the state of the component and progress through the entire framework.

Next to the topology manager there are 2 more core components, being the task distributor and the sink.

The task distributor is responsible for receiving tasks and distributing them to connected services. A typical source of tasks are scripts or UX (User Experience) elements, such as graphical user interfaces (GUIs), that submit processing tasks to this service. Therefore it is a single point of entry with a fixed API for everyone to use, greatly simplifying the use.

The last core component is the sink. It receives datasets that are submitted at the end of a pipeline. This does not mean that the submitted data is the final result since pipelines are broken down to the largest possible steps that avoids duplication of code or computations. An example of this is the calculation of the acquisition positions. Certain preprocessing steps need to be performed before the values should be used to reconstruct an image. These preprocessing steps are only performed once, by a dedicated pipeline, and the results made available via the sink to other pipelines.

As with the other 2 components, there is a simple API available to request datasets and a mechanism in place to get notified via events when certain datasets get updated.

3.2 Default pipeline components

As one might expect, there is no default layout of a pipeline. Hence, the framework supplies multiple default components with which one can build up a pipeline.

At the beginning of every pipeline there is a pipeline buffer component, which listens to the configured events that are broadcasted by the task distributor component and buffers them for the downstream component. The type of buffer can be altered depending on the needs between lifo, fifo, or priority.

The main goals of this component is the protection of the downstream components for overloading while also providing a metric that allows other components to automatically scale depending on demand.

The worker component does all the heavy lifting of the frame work and thus needs to be easily scalable. The datasets it needs can be configured statically or dynamically depending on the metadata that is included inside the received task. An important part of the framework is a standardisation of the data that flows between the different components, allows others to provide the actually processing algorithm while not having to worry about how to get the data nor how to actually implement a data pipeline.

The data sorter is most of the time the downstream component of the worker component. The datasets coming from the worker components can be out of order due to the parallel nature of the framework and thus need to be sorted in the correct order again. For this reason each acquisition is tagged with an acquisition index.

The sink component most often is the downstream component of the data sorter, completing our basic data pipeline structure, depicted in Figure 1.

3.3 Data writing and visualisation

The next logical step for the data collected on the sink component would be visualisation. The graphical representation of the data is a complex matter and highly dependant on what the end users wants. Therefore it is not included it the framework. Several components were developed to accommodate the loading of datasets in other software projects such as ImageJ (of Health (2024)).

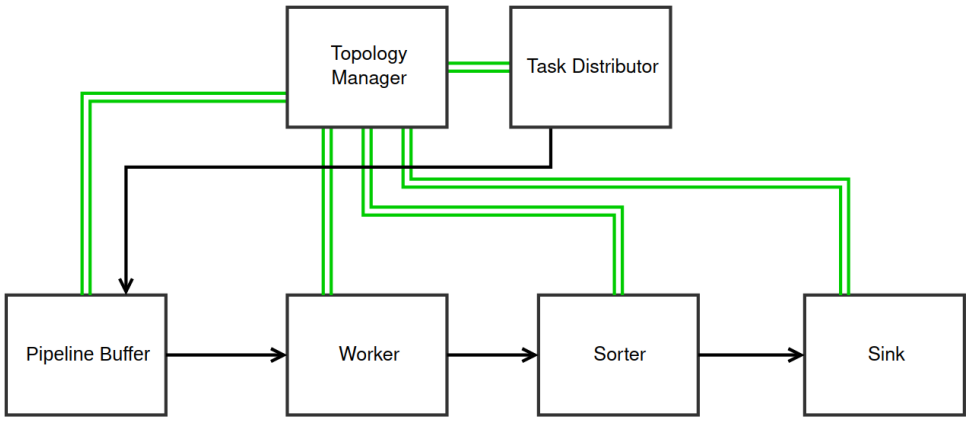

Figure 1: A block diagram representing the pipeline described in section 3.2. The green lines represent the connections with the topology manager, where the black arrows represent the data flow.

The most basic components are the data writers that output the content of the sink to HDF5 files which then can be opened with for example the silx viewer (Facility (2024)). Two other data formats are supported, jpg, to quickly view the results in the file browser of most operating systems, and tiff to enable more advanced visualisations using common packages like ImageJ.

For online visualisation at the beamline, the data is streamed to the Lavue viewer (Elektronen-Synchrotron (2024)), shown in Figure 2. The available datasets are provided dynamically, allowing the UX to adapt to the changes the users make to the configurations of the processing pipelines.

4 The deployment

It is advised to leverage the capabilities of the framework and thus split up individual data sources over individual data pipelines for the sake of scalability and the added benefit of the readability of the topology of the entire system.

Since the framework is based on components that are configured via yaml files, it can be deployed in a wide variety of ways. Just to mention a couple:

- Scripts;
- Containers;
- Kubernetes (Kubernetes (2024));
- Tango (J-M.Chaize et al. (1999));
- Slurm (SchedMD (2024)).

The approach will depend on many factors such as ease of use, available hardware, use case, and knowledge of the involved people.

4.1 Framework deployment at the Hard X-ray Micro/Nano Probe P06, PETRA III, DESY

The deployment at the Hard X-ray Micro/Nano Probe P06, PETRA III, DESY shows the flexibility of the framework well by making a balance between performance and ease of use. Components that are not data or compute hungry are all run at the beamline as tango servers. The reasoning behind this choice was to provide the beamline scientists a known means to start and stop data pipelines depending on their needs and enable control when something is not going as expected during periods of restricted support by the controls scientist.

The more demanding components are run on the Maxwell compute cluster located at DESY and scaled and managed via slurm. The slurm job starts a singularity container (Sylabs (2024)), which loads the required components, fetches a task from the upstream component and terminates again after processing and sending the results to the downstream component has finished.

The currently implemented data pipelines provide processing of:

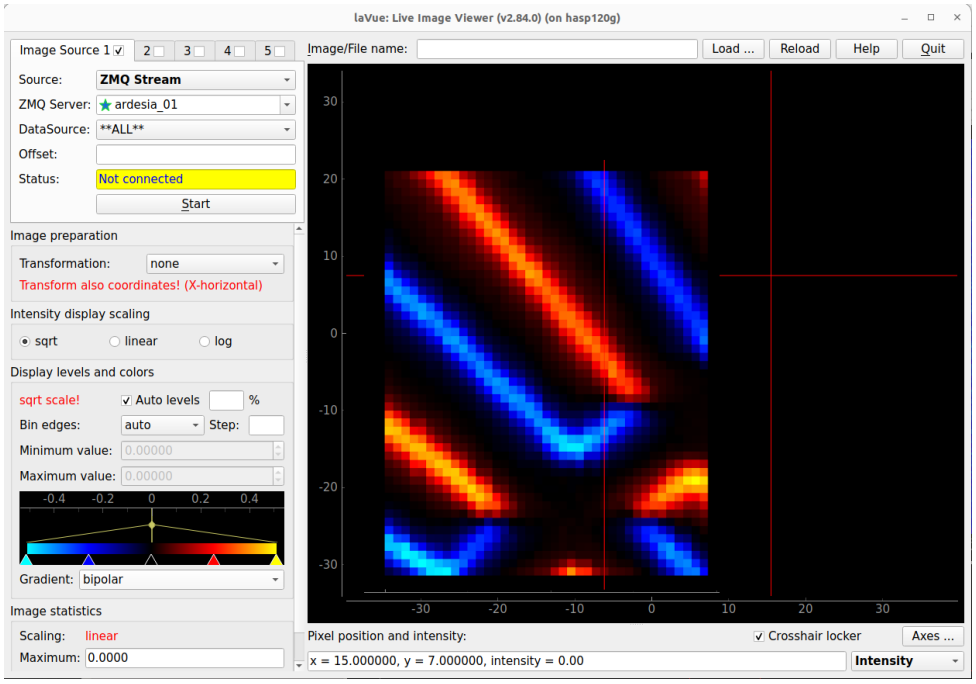

Figure 2: The lavue viewer showing online processed data received by a ZMQ data stream.

- Stage encoder positions;
- Interferometer positions;
- Counters;
- X-ray fluorescence (XRF);
- X-ray Diffraction (XRD);
- X-ray Beam Induced Current/Voltage (XBIC/XBIV);
- Differential phase contrast;
- Ptychography.

All data pipeline workers make as much as possible use of other software packages that are freely available. The XRF workers are based on the core fitting routines provided by the PyMCA software package (Solé et al. (2007)), where the XRD integration is based on PyFAI (Kieffer and Wright (2013)). This allows the users to create calibration files with their known software package and lowers overall development and maintenance time.

Workers are scaled on the HPC cluster to match the requirements. The compute cluster has grown over time and is therefore heterogeneous, an exact specification of a compute node can thus not be provided, but a typical CPU node has 2 socket and 512 GB of RAM, where a GPU node adds at least 1 GPU, being an NVIDIA P100, V100, or A100. The largest recorded scaling of the data workers has been seen for XRF and ptychography to 80 and 50 compute nodes respectively, perfectly showing the scaling capabilities of the framework.

The configuration of the processing settings is done via yaml files that are loaded and submitted via a graphical user interface to the metadata server. When scans are started and ended, the metadata is submitted to the metadata server, making the metadata available to all data pipeline components. Depending on the detector, events are generated for a certain number of streamed frames or a data file becoming available for processing.

Since all the data is processed based on the provided metadata, the data is ready to fulfil the requirement for the FAIR[1] principle.

[1] https://force11.org/info/the-fair-data-principles

5 Online control feedback loop

The event system used in the presented framework is also used by some key components of the beamline control system, making it really easy to close the loop. Although at the moment the online data processing has no means of altering a running scan, it can add or alter consecutive scans. The power of closing the loop between data acquisition and data processing has already been showed by Kourousias et al. (2023), Kandel et al. (2023), and others.

6 Outlook

The presented framework, meant to generalise data pipelines, has been successfully implemented at the Hard X-ray Micro/Nano Probe P06, PETRA III, DESY and has been appreciated a lot by both beamline scientists and users. Since everything is driven based on the automatically generated metadata, there is little room for error, increasing the usability by novel users, whom most of the time not even touch the processing settings.

The modularity of the framework makes it easy to port it to other instruments or add modalities. Of course nothing is perfect, neither is this framework. Some components are still missing, such as a fence component, similar to a fence used in OpenCL/CUDA. Initiatives have already started to use the framework to automate more operations at the beamline P06, such as automatic alignment of the focussing optics.

7 Acknowledgements

We acknowledge DESY (Hamburg, Germany), a member of the Helmholtz Association, for the provision of experimental facilities. Parts of this research were carried out at the Hard X-ray Micro/Nano Probe P06, PETRA III. This research was supported in part through the Maxwell computational resources operated at Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany.

References

Apache. Airflow. 2024. Airflow. Accessed here: `airflow.apache.org`.

Dask. Dask. 2024. Dask. Accessed here: `https://www.dask.org/`.

Deutsches Elektronen-Synchrotron. Lavue. 2024. Lavue. Accessed here: `https://github.com/lavue-org/lavue`.

European Synchrotron Radiation Facility. Silx-kit. 2024. Silx-kit. Accessed here: `http://www.silx.org/`.

J-M.Chaize, A.Götz, W-D.Klotz, J.Meyer, M.Perez, and E.Taurel. Tango - an object oriented control system based on corba. 1999.

Saugat Kandel, Tao Zhou, Anakha V. Babu, Zichao Di, Xinxin Li, Xuedan Ma, Martin Holt, Antonino Miceli, Charudatta Phatak, and Mathew J. Cherukara. Demonstration of an ai-driven workflow for autonomous high-resolution scanning microscopy. *Nature Communications*, 14(1), September 2023. ISSN 2041-1723. doi: 10.1038/s41467-023-40339-1.

J. Kieffer and J.P. Wright. Pyfai: a python library for high performance azimuthal integration on gpu. *Powder Diffraction*, 28(S2):S339–S350, September 2013. ISSN 1945-7413. doi: 10.1017/s0885715613000924.

George Kourousias, Fulvio Billè, Francesco Guzzi, Matteo Ippoliti, Valentina Bonanni, and Alessandra Gianoncelli. Advances in sparse dynamic scanning in spectromicroscopy through compressive sensing. *PLOS ONE*, 18(11):e0285057, November 2023. ISSN 1932-6203. doi: 10.1371/journal.pone.0285057.

Kubernetes. Kubernetes. 2024. Kubernetes. Accessed here: `https://kubernetes.io/`.

National Institutes of Health. Imagej. 2024. Imagej. Accessed here: `https://imagej.net`.

OrangeDataMining. Canvas. 2024. Canvas. Accessed here: `orangedatamining.com`.

SchedMD. Slurm. 2024. Slurm. Accessed here: `https://slurm.schedmd.com/`.

V.A. Solé, E. Papillon, M. Cotte, Ph. Walter, and J. Susini. A multiplatform code for the analysis of energy-dispersive x-ray fluorescence spectra. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 62(1): 63–68, jan 2007. doi: 10.1016/j.sab.2006.12.002.

Sylabs. Singularity. 2024. Singularity. Accessed here: `https://docs.sylabs.io/guides/latest/user-guide/`.

ZeroMQ. Zeromq. 2024. Zeromq. Accessed here: `https://zeromq.org`.