Goals for Shared Scattering Science Pipelines[1]

Ray Osborn* and Justin M. Wozniak**
*Materials Science Division, Argonne National Laboratory
**Mathematics and Computer Science Division, Argonne National Laboratory

The ongoing development of high-resolution, high-frame-rate detectors and the x-ray and neutron scattering science capabilities they offer has created a computing challenge. Large experimental datasets must be rapidly stored and managed for near-real-time analysis. The construction of analysis *pipelines* partially automates the analysis process, transferring data to storage and among multiple analysis software packages. In this white paper, we describe initial efforts in pipeline development, identify areas for cross-institution collaboration, and pose some open questions.

The Pipeline Endpoint

The goal of any pipeline is to reduce the data to a form that is suitable for scientific visualization, modeling, interpretation, and publication. However, reconstructing an image or transforming scattering data to S(**Q**) in an automated pipeline is only the first stage of a scientist's interaction with the experimental results. After that, it is necessary for the scientist to interrogate the products of this pipeline in ways that are rarely possible to predict in advance. It might be necessary to fit either phenomenological or first-principles models to sections of the data, compare and/or scale different datasets that differ by some parametric variable, or perform a variety of statistical tests on either selected regions or the entire data volume. The choices a scientist makes in performing this post-pipeline analysis are usually specific to the particular scientific question being addressed by the experiment, and therefore impossible to encapsulate in a predefined pipeline.

It is as important therefore that we have a plan for ensuring that the results of any pipeline are accessible in a convenient and reasonably standardized form as it is to develop tools for constructing the automated pipelines in the first place. If the results of data reduction pipeline reduce the data to a size that is manageable by conventional resources (*e.g.*, desktop or laptop computers), then this is not a significant issue for the facilities; it is merely necessary to give the scientists a means of downloading the reduced data. However, if the volumes of reduced data are still comparable to the original raw data volume, then new paradigms for accessing the data need to be developed. It is no longer sufficient to deposit the data in an archive, if the user if then required to download it in order to perform any follow-up operations. We have to develop a network architecture that allows fine-grained access to the data without requiring significant data transfers.

---

[1] This work is a collaboration with Stephan Rosenkranz, Matt Krogstad (MSD), Guy Jennings (APS), Michael Wilde, Ben Blaiszik, Kyle Chard, and Ian Foster (MCS).

We have been developing a prototype of a combined pipeline/post-pipeline architecture in the context of the Argonne Grand Challenge LDRD on Data Driven Science, *Discovery Engines for Big Data*. In particular, we have performed a number of experiments to measure single crystal diffuse scattering using fast-area detectors, such as the Pilatus 2M and 6M detectors, that can generate data rates of several GB/min. For example, three-dimensional measurements of S(**Q**) over the entire phase diagram of a compound (*e.g.*, 6 samples x 25 temperatures) can produce 5 TB of data in one or two days. In our particular pipeline, raw images in TIFF or CBF format are streamed to a remote server and automatically stacked in NeXus/HDF5 files by Python scripts, which also harvest the relevant instrumental and sample metadata. These are then subjected to a series of automatic data reduction steps to refine the crystal orientation parameters and transform the data from instrumental coordinates to reciprocal space coordinates. This pipeline is controlled by Swift scripts running on the remote server. Swift is a data workflow language designed to facilitate the use of high-performance computing resources without the need to write elaborate MPI programs.

In order to give the scientist access the data after this automated pipeline has been completed, the reduced data are registered in the Globus Catalog, which stores both the location of the server containing the data and the file paths to the data on the server. In this way, the data are immediately available for remote visualization and analysis, including real-time analysis during the experiment, through a Python Remote Object (Pyro) server that accesses the data identified through the catalog irrespective of its location.[2] This is typically used during an experiment to refine parameters before launching automated workflows, as well as for analysis after the workflows are complete. All of the data and metadata are accessible through a simple Python API that loads entire metadata trees without transferring large data volumes. The API can be invoked from a Python shell or using an extensible Python GUI, NeXpy (http://nexpy.github.io/nexpy). Subsets of the data can be transferred as Numpy array slabs, but the design will allow for many of the operations to be performed on the remote server, with the assistance of Swift scripts to parallelize larger operations.[3]

We have made widespread use of HDF5 external links in the prototype as a way of separating the raw data, that should be immutable and protected from corruption, and the higher-level metadata both from the experiment and added through the pipeline. We believe the system could be improved by defining global URIs for the raw data files, defined through the Globus Catalog, which will allow the high-level "wrapper" files to access the data from anywhere. Different scientists, who may be performing different modes of analysis on the same data, *e.g.*, powder diffraction or PDF analysis, can copy these wrapper files, which are typically only a few MB in size, from the data servers and customize them for their particular application, with each able to access the raw or processed data using the Python Remote Object protocol.

[2] Wozniak et al. Big data remote access interfaces for light source science. Big Data Computing 2015.
[3] Wozniak et al. Big data staging with MPI-IO for interactive X-ray science. Big Data Computing 2014.

This prototype has been successfully demonstrated at both the APS and CHESS, and will soon be tested for joint analyses of diffuse scattering at APS and the Spallation Neutron Source. We believe that the experience gained on this project should be of use in encouraging inter-facility cooperation through the standardization of remote data access procedures, and, potentially, in the design of remote data facilities.

Need for shared technologies

Multiple opportunities exist for technology sharing, with multiple end goals. The use of common tools eases data sharing and collaboration, as scientific techniques and results are more easily understood. The availability of common tools would improve beam time productivity as well-understood, well-tested, and well-maintained codes are ready for use.

The prototype that we have been exploring would benefit considerably from integration with other projects. As an example, we would be interested in integrating the Brookhaven Data Broker into the NeXpy GUI, which is currently designed to open HDF5 files. While NeXpy has importers for other forms of data, such as TIFF or SPEC files, it would be valuable to make the process of inputting the data more transparent to the user by making it work with a standard Open File dialog. This should not be hard because, ultimately, both NeXpy and the Data Broker convert external formats into internal Numpy arrays. By integrating the two, it would be possible to extend the Pyro protocol to use the Data Broker to allow other formats to be input through the Globus Catalog. Conversely, we would be interested in incorporating the NeXus Python API into the Data Broker package to make the reading of NeXus files more generally available through other channels.

A second level of integration would be to use the NeXpy plugin architecture to add menus that directly access functions in the scikit-xray package or other software packages, where this makes sense. The NeXpy GUI can be customized for specialized applications, often without extensive knowledge of PyQt.

Data manipulation: The community is already converging on the common practice of expressing high-level computational tasks in Python, with the critical data type being Numpy arrays backed by NeXus/HDF5 data files. Interoperability can be achieved through a mix of information exchange and typical software engineering. The exchange of information among user groups and institutions is critical to identify and promote tools and techniques that offer the greatest productivity boosts. Software engineering best practices must be applied to support the code sharing process. This includes refactoring and cleaning, documentation, version and release management, distribution, installation, and maintenance.

Pipeline construction: The community is already moving toward Python for sequential processing and basic data manipulation. Performance-critical sections are expected to be written in C, C++, or Fortran, and exposed to Python. Such libraries are likely to be usable in other settings, such as high-performance computers.

The Swift workflow language is a promising option for multi-node scalability. It offers excellent support for calling into Python (via the command line or Swift's embedded Python interpreter).[4] It also runs on a wide range of resources, from laptops to clusters to high-performance computers.

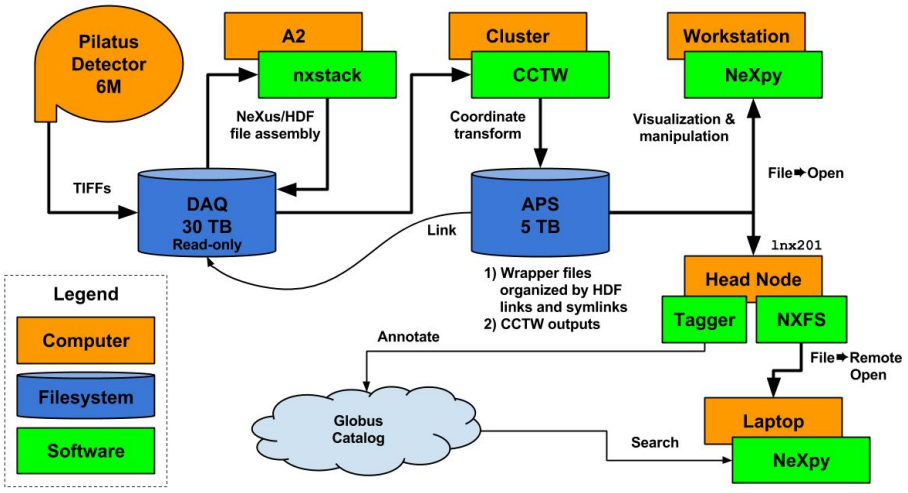
Pipelines involve automated and interactive phases. It is unlikely that a pipeline can be fully automated, as this implies that the scientific problem has been solved. This mix of pipeline modalities requires the ease of reusing existing pipeline fragments and component tools, and thus requires high-quality, easily-usable software. User groups must be able to rapidly construct pipelines from such components for an experiment session to obtain high-quality computational results along with high-quality experimental results.

We envision that users will be able to select from existing or custom Python-based programs and libraries and well-curated Swift workflows and paste together a scalable solution.

Data access and sharing: Given highly understandable results from high-quality pipelines, data must be easily shared. This involves data cataloging, data transfer, and remote object access.

Catalogs are useful when they are easy to browse and appropriate metadata conventions are in place. Globus Catalog offers web browser, Python API, and shell tools. It offers a fully generic metadata scheme that could be used by any scientific domain. Its catalog entries can be “empty” or point to data accessible for Globus Transfer. We have also used it to point to other data sets by extending the use of metadata fields.

Case Study



[4] Wozniak et al. Interlanguage parallel scripting for distributed-memory scientific computing. WORKS 2015.

Our recent run at Cornell CHESS illustrates many of these components in operation. Over a week long run, the detector wrote 21 TB of raw TIFF images to the Data Acquisition (DAQ) file system. The images were automatically stacked into 3D NeXus files by a process that monitors new files. The data is then cataloged and made available for viewing in NeXpy (over remote X Windows or via remote object access). These datasets were transformed by CCTW, which was parallelized by Swift running on the Cornell CLASSE cluster, reducing transform time from 40 minutes to just over 5 minutes.

Open questions

How can we bring experiment and HPC together? Initial attempts to connect the APS to ALCF resources have been promising but are not yet a regular occurrence. More prototypical examples of experiments that can utilize both systems well must be sought out.

Stream processing and co-scheduling. Utilizing HPC will probably require efficient streaming preprocessing steps to rapidly reduce data sizes. In our runs at CHESS, simple compression reduced data size by 95%, yet this had to be performed *after* data was resident on disk. Such operations must be performed as close to data acquisition as possible, so that the transfer to the HPC resource is not a bottleneck. In some cases, it may be a performance benefit to transfer data directly to the HPC resource without staging from disk, which will require co-scheduling or reserving time for experimental use on the HPC system.

How can we expose pipeline construction to end users? Each experiment is different. Workflows must be reconfigurable from their core components, and extensible with novel components. This should be investigated with the use of appropriate workflow technologies and high-quality software engineering techniques.