Big Data Remote Access Interfaces for Light Source Science

Justin M. Wozniak,*† Kyle Chard,† Ben Blaiszik,† Ray Osborn,‡ Michael Wilde,*† Ian Foster*†§
*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
†Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
‡Materials Science Division, Argonne National Laboratory, Argonne, IL, USA
§Department of Computer Science, University of Chicago, Chicago, IL, USA

***Abstract*—The efficiency and reliability of big data computing applications frequently depend on the ease with which they can manage and move large distributed data. For example, in x-ray science, both raw data and various derived data must be moved between experiment halls and archives, supercomputers, and user workstations for reconstruction, analysis, visualization, storage, and other purposes. Throughout, data locations must be tracked and associations between raw and derived data maintained; data accesses, even over wide area networks, must be highly responsive to allow for interactive visualizations. We use here a typical x-ray science workflow to illustrate the use of two recently developed techniques for data movement, archiving, and tagging. The first is a scalable catalog for referencing, managing, and performing remote operations on distributed data. The second is a novel remote object interface for structured (HDF-based) data set manipulation and visualization. Our description of these techniques and their application to x-ray science problems sheds light on big data problems in experimental science, the gap between conventional big data solutions and scientific requirements, and ways in which this gap may be bridged.**

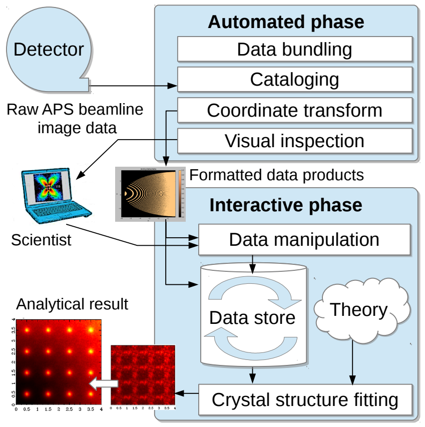


Figure 1: APS data workflow, showing the *automated* data ingest and *interactive* data examination phases.

I. Introduction

Synchrotron x-ray facilities, such as the Advanced Photon Source and Advanced Light Source, use a variety of scattering, imaging, and spectroscopic techniques to address a range of problems in materials science. The increasing brightness of such x-ray sources, coupled to recent developments in detector technologies, means that they must handle significantly greater data rates than in the past, both on individual beam lines and across facilities as a whole. Pulsed neutron sources, such as the Spallation Neutron Source, face similar big data challenges; their instruments include large multidetector arrays that require complex data reduction.

Access to such national scientific resources is granted after a competitive review process, with researchers often waiting months for a few hours or days of experimental time. Delays in data analysis can make all the difference between a successful and failed experimental session. Thus, techniques that can enable more rapid and effective (and, in particular, real-time) analysis can greatly enhance the productivity of both individual researchers and these large-scale scientific investments.

Single crystal x-ray scattering experiments provide a good illustration of the challenges associated with experimental data analysis. A focused beam of x-rays is scattered by a small crystalline sample into a large pixelated detector. A typical detector collects images of size 2,048×2,048 pixels at around 10 Hz and thus can generate data at 160 MB/s, but frame rates of over 100 Hz are now possible. Given inevitable pauses for experiment setup, and so forth, a week-long experimental campaign may produce 10 TB of raw data.

Such large data sets must be subjected to both automated and interactive analysis. Some data reduction tasks are common to many scattering experiments, and others may be unique to a single experiment. Software tools are required to transform raw data from instrumental coordinates to the reciprocal lattice coordinates defined by the crystal structure. Sharp Bragg peaks that are periodic in the reciprocal lattice can be used to determine the average crystalline structure. In disordered materials, such as those used for batteries or electronics, the real structure deviates from this average, producing substantial *diffuse scattering* between the Bragg peaks. Computational techniques exist to fit potential defect structures against this experimental data through simulation [1]. Those techniques require optimization over large data volumes.

Here we treat single-crystal x-ray scattering as a big data problem and present techniques for organizing both raw experimental data and subsequent derived data and for managing and interacting with such large data over networks. We show how these techniques can accelerate real-time analysis of x-ray scattering data and thus improve the effectiveness of large-scale scientific investments.

Our approach to x-ray data organization and access is based on the construction of ad hoc *data lakes* for x-ray science. A data lake, in contrast to a data warehouse, 1) retains all data in its natural state [2] so that 2) data requirements can be defined at query time [3]. In so doing, it avoids the need for extensive preparatory extract-transform-load (ETL) processing before data can be accessed, at the expense of potentially more complex data accesses subsequently. The data lake approach is appropriate for x-ray science because we need to enable rapid access to data as soon as it is generated and to support a wide variety of analysis methods.

As illustrated in Figure 1, the data processing supported by our data lake implementation thus comprises two phases. The simple preparatory (*automated*) phase performs only processing essential to the creation of the data lake, with the goal of making data available to users as soon as possible. The *interactive* phase involves direct user interaction with the data lake's contents. Here, we implement powerful remote abstractions and software interfaces to maximize scientific utility without bulk data movement

This paper has two key generalizable contributions to the construction and use of data lakes for x-ray science. First, we describe a scalable catalog model that provides the basis for tracking the provenance, status, and structure of all data loaded into the data lake. Second, we describe a remote object interface for HDF5 data files that permits easy and efficient interactive access to even extremely large datasets over commodity Internet connections. This interface allows materials scientists to manipulate big data sets from the experiment hall, from the office, or from home without bulk data transfer or copy. The interface provides the application-specific 'smarts' to make the bulk data lake useful.

The remainder of this paper is organized as follows. In §II, we describe the scientific problem in more detail. In §III, we describe the catalog used to manage data. In §IV, we describe the architecture of our remote object interfaces for big data. In §V, we consider related work. In §VI, we evaluate the techniques from a performance perspective. In §VII, we offer summarize our conclusions. In §VIII, we discuss future work.

II. Scientific overview: x-ray diffuse scattering

We illustrate in Figure 2 a scientific workflow that motivates this work. This workflow spans systems at the Advanced Photon Source (APS) at Argonne National Laboratory (ANL), the Argonne Leadership Computing Facility (ALCF) at ANL, and the cloud-hosted Globus Catalog service [4].

At the APS (1), detector data is collected at a beamline station, a lead-lined hutch containing the sample, area detector, and a beamline computer [5], typically a PC with a specialized I/O interface to the detector hardware. This PC must not be delayed by any work other than transferring detector data to storage; if its buffers are filled, it will drop detector images. In our architecture, data is stored in a RAM-based filesystem, so storage is highly constrained. Data must be transferred from this computer to a larger hard disk-based storage system. We

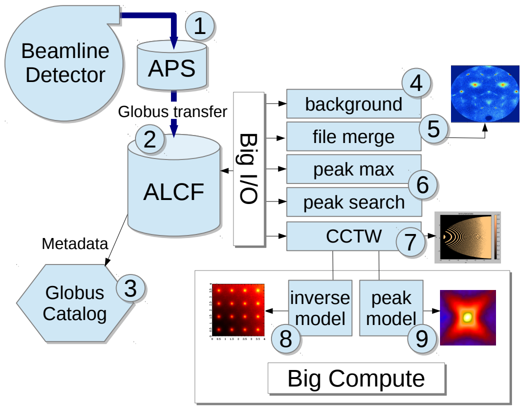


Figure 2: Overview of APS beamline workflow.

do so by running a lightweight `rsync`-based script at a high `nice` level (that is, low priority) and deleting the data once it has been safely transferred.

Data sizes and other parameters vary from experiment to experiment. In our most recent use of the APS beamline facility, in April 2015, scattering data was collected as a sample was rotated 360 degrees in tenth-of-a-degree increments, yielding a total of 3,600 images. Each image comprises 2048×2048 32-bit pixels, (i.e., 16 MB) and thus a full dataset is 56 GB. (This high resolution is required in order to capture the subtle signals associated with the sample's structural defects.) Images are produced at a rate of 10 per second and thus at a peak rate of 160 MB/s. Following the collection of a complete 3,600-image dataset, which takes about 10 minutes, sample conditions may be changed (e.g., by changing temperature), and the process is repeated. Or, a new sample may be substituted. The former process can be fully automated, allowing data collection to proceed 24/7 during the experimental window. Overall, data collection rates averaged 14.5 MB/s during our last one-week session (April 2015).

Once on ALCF resources (2), automated data-intensive operations (4)–(7) perform data bundling, metadata creation, and cataloging quickly and automatically. Background noise from the detector, determined via an earlier calibration process, is subtracted from the data (4). Each rotation of 3,600 image files is then assembled into a single NeXus-format file [6]. NeXus is a set of conventions for the use of the HDF5 structured data format [7]. Each file is tagged with appropriate metadata such as wavelength, temperature, and other instrumental and sample parameters.

We also store this metadata and file location (host and physical path) in the Globus Catalog to enable browsing, search, and inspection through the web browser or the NeXpy GUI [8]. NeXpy recently was extended to allow users to browse Catalog entries and view remote data (§IV). This feature makes it easy for scientists to check for experimental errors while in the experiment hall, allowing for detection of calibration errors or other anomalies that could, after the fact, invalidate an entire campaign [9].

Automated data analysis operations are then performed. A maximal value search is performed over all data to provide input to the peak search algorithm that is used to find all Bragg peaks in the experimental data. (These peaks span the 2D image files in 3D.) These locations are then fed into a coordinate transform that requires significant computation and data movement.

The transformed data, which has been mapped onto the sample's reciprocal space coordinates, can be used for two purposes, both requiring large computation. An inverse modeling scheme (8) is used to compare against simulated crystal structures, iteratively converging on a good approximation to the true crystal structure, including defects. This scheme employs an evolutionary algorithm to evaluate a large "population" of simulated crystal structures by running them through a forward model that produces the simulated scattering image for that structure [10].

III. Cataloging distributed big data

A single x-ray dataset may contain thousands of files, distributed over several locations and at various stages of analysis. Simulations and analyses may further create additional files that are meaningful only in relation to the raw data from which they are derived. Thus, the task of managing and performing operations on large distributed datasets is both difficult and time consuming. To simplify this task we have developed a scalable cloud-hosted data catalog that allows users to define "datasets"—collections of remote data (files, directories) with associated metadata. The catalog provides a location-independent view of datasets from which users can manage, describe, query, and perform actions on (e.g., analyze, transfer, share) as a single abstract entity. Thus, operations can be aligned with scientific activities rather than physical storage resources.

A. Globus Catalog

Globus Catalog is a cloud-based service that allows users to create and manage datasets. Datasets exist within the context of a catalog — a user-defined namespace for grouping related datasets (e.g., for a research group, project, or user). Catalogs contain one or more datasets; datasets contain references to one or more members (remote files or directories); and metadata (in the form of key-value annotations) may be associated with members and datasets. Figure 3 provides a high-level view of Globus Catalog.

Globus Catalog provides a level of abstraction from underlying and potentially distributed data. Rather than working with individual files and directories that are locally hosted, users can group data across storage repositories, compute resources, and even cloud storage and then perform remote operations on them such as browsing, annotation, and sharing. Since Globus Catalog is designed for collaborative use, it has features to support independent access-controlled views of the same underlying data. Catalogs and datasets can be shared

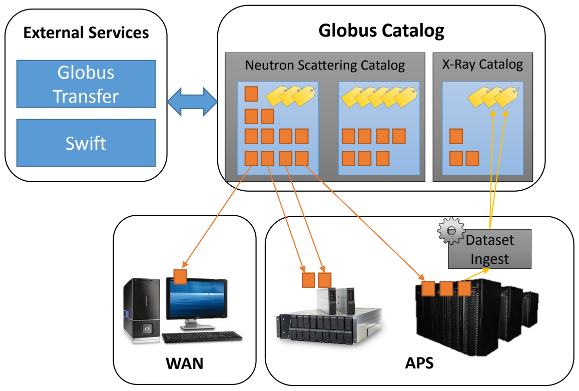


Figure 3: Globus Catalog showing two catalogs (Neutron Scattering Catalog and X-Ray Catalog). Neutron Scattering Catalog has two datasets with a number of members (files hosted on different resources within APS and externally) and annotations on each. X-Ray Catalog has several members as well as annotations automatically extracted from files hosted on the rightmost APS compute resource. Services and tools that interact with the catalog are shown on the left.

with groups of users, allowing specified control of read/write operations on their contents.

Globus Catalog is hosted as a highly available cloud-based service. A single instance of the service is operated for all users; and its self-service interfaces allow users to create, customize, and manage personalized collections. All capabilities are exposed via an intuitive web interface as well as programmatic REST APIs. We have also developed client libraries and a command line interface for interacting with catalogs.

B. Referencing remote data

Globus Catalog relies on the remote access mechanisms provided by Globus [11] to reference and perform remote operations on distributed data. While Globus is most well known for providing high performance and secure data transfer, synchronization, and sharing, it also implements a standard model for addressing and securely accessing distributed storage.

Globus exposes a network of over 8,000 registered "Globus endpoints"—logical representations of storage resources that implement Globus data access APIs. These APIs provide a common interface to storage systems that may be deployed on various operating systems, architectures, and file systems, (for example, Linux, MacOS, Windows, HPSS, and even Amazon S3 cloud object storage). Systems can be made accessible to Globus by installing Globus Connect, a lightweight agent that exposes Globus security and access APIs. Currently, Globus relies on the GridFTP protocol [12] to perform remote operations on data; however, it will soon also support secure HTTP access.

In the catalog, remote data ("dataset members") are registered with a uniquely composed URI that references a file or directory with respect to its Globus endpoint name and the path within that endpoint. These URIs both identify the data and enable operations to be performed on that data remotely. For example, data member URIs can be retrieved and passed to the Globus transfer service to move or synchronize data, or by a Swift script to download and process data.

Globus Catalog also enables remote data to be shared with other users. To provide this capability we build upon the Globus shared endpoint model [13]. Here, shared endpoints are created by defining a mapping to a specific path within an existing Globus host endpoint. The shared endpoint abstracts this mapping via a new endpoint name that can be accessed in the same way as a host endpoint. However, rather than using host endpoint access control mechanisms–which typically authorize access based on local user accounts–shared endpoints instead delegate access control decisions to the Globus service. Read and/or write access permissions may be assigned to files and directories contained within a shared endpoint. As shared endpoint access control lists (ACLs) are managed by the Globus service they are evaluated when accessing data and may be modified (e.g., revoked) at any time with changes reflected immediately. Thus, any data referenced by a cataloged dataset member and contained within a shared endpoint can be shared via the Globus APIs. When Globus Catalog users choose to share such data, Globus Catalog forms a request to Globus to update data access ACLs accordingly for all contained members.

C. Flexible annotation

An important aim of Globus Catalog is to support flexible user-oriented annotation. Annotations provide a way for describing datasets and their members; they may be used for managing datasets, for understanding datasets when shared, and for conducting future search and discovery. Rather than impose strict metadata schema and vocabularies, Globus Catalog instead allows users to define and associate their own typed metadata with datasets and members. When defining an annotation, the user needs only to select a unique name and then describe its type (string, int, float, etc.), multiplicity (single or multi-valued), and uniqueness. The new annotation along with its value (or values) can then be associated with a dataset or member. Catalogs and the datasets and members they contain do not share the same annotation namespace; hence, they avoid collisions across catalogs and allow annotations to be interpreted differently depending on context.

The flexibility of our annotation model leads to significant implementation challenges. User-oriented tagging models typically result in sparse data, since researchers tend to characterize data in varied ways, and only a few annotations may be applied to any one item. Given the potentially huge range of attributes that can be applied and the requirement to support rich search across these attributes, researchers are actively investigating scalable, secure, and efficient models. To address these challenges, we built on the Tagfiler [14] tagging system. Tagfiler imposes a decomposed Entity-Attribute-Value (EAV) model that allows tags of the type (subject, tagname, object) to be efficiently stored. Each tag is stored in a separate table that enforces tag constraints (e.g., typing). This model is optimized for tag insertion and query but is not well equipped to handle storage and retrieval of many tags.

D. Automated metadata extraction

Much valuable metadata is locked within various data formats. For example, in x-ray scattering, the widely used NeXus and other HDF5 formats include metadata describing sample materials, experimental protocols, investigators, and other information important for understanding the underlying data. On some occasions, important metadata is written to a co-located text file or even encoded within file and directory names. Often this metadata is represented in proprietary formats and using various ontologies and vocabularies. Associating such information with datasets and members in a catalog provides enhanced management and search capabilities for users. Doing so in an automated way reduces user overhead, increases utilization, and improves the value of the catalog.

Since much of our x-ray scattering data is contained within NeXus files, we have developed a NeXus parser to extract common NeXus-defined metadata such as metadata describing instrument components and state (e.g. source, beamline used, collimator, attenuator, detector); the specific sample (e.g., sample dimensions, material composition, and processing description and history); data and compute resource locations; and links to plottable data from NeXus HDF5 files. We have implemented this parser as a lightweight Python script that leverages the Globus Catalog APIs to automatically register metadata with cataloged files.

E. Querying and discovering data

Once data is effectively cataloged, we aim to support browsing and discovery of the data. Researchers are increasingly accustomed to search-based discovery methods (e.g., Google, Google Scholar, Web of Science). Given the amount and wide variety of data and metadata that could potentially be cataloged, we focus on supporting intuitive query methods such as free text search, range queries, and faceted search. Free text search enables partial text matches and rankings based on text importance. For example, users can search for datasets containing the word "tomography" or the name of a particular material.

Range queries may be combined with free text searches to filter the results to a particular subset of search results. For example, a researcher may be interested in finding "tomography" datasets for which $8\ \text{keV} < \textit{photon energy} < 10\ \text{keV}$.

Faceted search [15] is an intuitive technique for exploring classified or categorized information such as consumer products in online retailers. It is especially effective for large amounts of data because it both provides a summary of the data and allows drill down via increasingly specific filters. Globus Catalog provides customized faceted views for users based on the metadata available to them in a specific catalog.

These query methods allow users to find datasets and members quickly based on their associated annotations. They can combine free text and faceted search results to explore their data across different dimensions. They can apply increasingly precise queries and complex combinations to further filter search results. Results are ranked based on the proximity of the match, and only datasets accessible to users are shown.

F. Supporting remote operations

Many tools and services have been designed with an implicit assumption of single and/or local file access. In domains such as x-ray science, such assumptions are no longer tenable. Rather, researchers must often interact with large collections of distributed data. The Globus Catalog dataset abstraction provides a framework not only for organizing and managing such distributed data collections but also for implementing remote operations on this data. External services can use the Globus Catalog and Globus data access APIs to leverage the structured dataset representation to perform remote operations.

The first such example of a remote operation is the ability to transfer and share an entire or partial dataset. To perform this operation, we have integrated Globus Catalog with Globus transfer. When a user chooses to transfer or share a dataset, we use the Catalog APIs to find all members of a dataset and construct a series of transfer jobs for each location included in the dataset. The transfer jobs specify either the remote endpoint on which to transfer the data or a series of ACLs for sharing the data with other users.

A second example of a remote operation is the need to visualize and manipulate remote data. Such examples are common during the acquisition and analysis phase of an experiment as researchers interact with their data and modify their protocols and analyses. To address this need, we have modified NeXpy to execute queries within a catalog, and select dataset members for visualization from the resulting list. To provide remote access within the referenced file, we have leveraged Pyro4 (§IV-B).

A third example of remote operations is the need to analyze datasets. For example, users often want to execute a quality control algorithm on a dataset or process a particular dataset with a particular analysis routine. For this purpose we have extended Swift workflows [16], [17], [18] to be able to run analyses based on dataset references. The scripts operate by first retrieving the list of dataset members from the Catalog API and then transferring data (where needed) to the worker nodes on which the analysis is executing.

These operations demonstrate the potential for performing remote operations using Globus Catalog. However, many more operations may similarly benefit from such support. Activities such as data publication, long-term archival, replication, and many other types of analyses could all benefit from this approach. As future work, we intend to enhance other services to become dataset compliant, so that they can act on collections of distributed data.

G. Using the Catalog

Globus Catalog plays a central role in our x-ray science workflow. Specifically, all data that are acquired is first processed by customized NeXus HDF5 extraction scripts. For each batch of files, a new dataset is registered in the catalog, the files are registered as members of these datasets and extracted metadata is associated with each member. Aggregate metadata and fixed metadata describing the experimental samples and conditions are also associated with the dataset.

Other tools in the workflow (e.g., NeXpy and Swift) interact with data stored in the catalog dynamically. For example, analyses may be executed on abstract datasets, where the analysis routine is configured to determine file locations via catalog queries and then to transfer some or all of the data for analysis. During execution, workflows may update dataset members and annotations. New datasets may be created and references to raw data added in the form of special annotations.

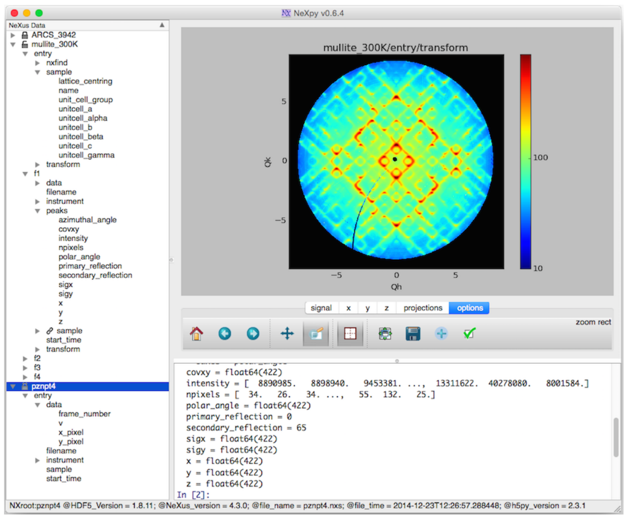
In our x-ray science scenario we have so far created five production catalogs that collectively contain 59 datasets, 1427 dataset members, 119 unique annotation definitions, and 12,259 user annotations.

IV. Remote interfaces for structured big data

We seek to enable convenient, efficient access to remote data sets to support visual data inspection, plotting, and scientific investigations. The NeXpy GUI tool integrates an IPython shell with matplotlib visualization and NeXus data access to create a full-featured environment for these tasks.

A. User workflow

Here, we describe requirements for the remote data access features based on user access patterns.



```
covxy = float64(422)
intensity = [  8890985.   8898940.   9453381. ...,  13311622.  40278080.   8001584.]
npixels = [  34.   26.   34. ...,   55.  132.   25.]
polar_angle = float64(422)
primary_reflection = 0
secondary_reflection = 65
sigx = float64(422)
sigy = float64(422)
x = float64(422)
y = float64(422)
z = float64(422)
In [2]:
```

NXroot:pznpt4 @HDF5_Version = 1.8.11; @NeXus_version = 4.3.0; @file_name = pznpt4.nxs; @file_time = 2014-12-23T12:26:57.288448; @h5py_version = 2.3.1

Figure 4: NeXpy screenshot.

1) Original usage: NeXpy originally could access only local data, which meant that users had to copy NeXus files from the data store to the local laptop: an expensive process since files are typically 50 GB. Once copied, the file would be found and opened using the traditional IPython File→Open widgets. Once open, the HDF data structure would be shown in the left pane as an expandable tree, as shown in Figure 4. (If multiple files were open, multiple trees were presented, allowing users to rapidly jump from one HDF variable to another.) Double-clicking on an HDF variable would plot the variable in the plot pane. Variables were available for manipulation automatically in the bottom pane. The user could also rapidly zoom in and out, rotate, and produce 2D projections from 3D data. The user could thus easily and rapidly access small slices of a large 3D dataset.

2) Room for improvement: Retrieving an entire file prior to opening, browsing, and analysis is unnecessary, given the GUI pattern. The expandable HDF tree widget implies that not all data or even metadata need to be retrieved immediately upon remote file open. Typical plots require only a slice of the underlying 3D data to produce a heat map or XY plot, in which case only that slice of data must be retrieved.

3) Remote usage: In the remote case, NeXpy usage starts by using the Globus Catalog widgets in the GUI to perform a metadata service. The user selects a catalog from the list managed by the cloud-based server. Then, the user enters key-value pairs to find the *dataset* of interest. A dataset contains multiple *members*, each typically a file or directory with associated key-value pairs. (Members were originally intended to represent files accessible through Globus data transfer, but these entries have been overridden in this work to refer to physical file locations. The host servicing the file is provided in metadata on the file.)

Have identified a file of interest, we want to allow a user to open it in NeXpy seamlessly, with minimal modification to NeXpy. We want good performance both for such remote access and for low-level IPython manipulation of HDF variables. Thus, we need to subclass the NeXus API on these variables so that these HDF-like operations are captured and replayed on the actual data on the remote host.

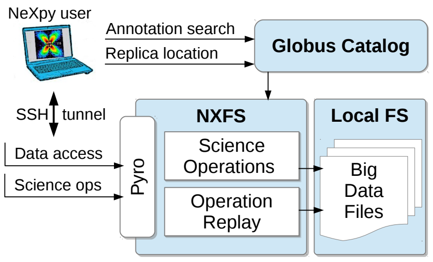


Figure 5: The NXFS system.

B. NXFS

Our solution, the NeXus File System (NXFS), is shown in Figure 5. Usage starts once NeXpy obtains a host/file of interest. NeXpy instantiates a Python object of type `NXFileRemote` (NXFR), a subclass of the original Python object `NXFile` representing a local file. Both classes support operations such as *open*, *close*, and so on, but the key NeXus API operations are the Python `__getitem__` and `__setitem__` methods, which implement the Python square bracket operators. For example, if `F` is a NeXus file, `F["g"]["v"]` would retrieve (multidimensional) variable `v` from HDF group `g` as a Numpy array. This concise syntax allows for easy data manipulation in the IPython pane.

To achieve remote access, we use the third-party, open source Pyro4 [19] package (for Python Remote Objects). Pyro allows the user to register an object `x` for remote access, returning a port. This port then listens for connections. A client process over the network may connect to this port, returning a proxy object `p` that apparently supports all of the methods that `x` did; thus, from the client perspective, `p` *is* `x`. The methods of `p` are populated with the methods of `x`, and may be accessed directly by the client. Thus, unmodified code may be passed a proxy object, as was done in the performance study (§VI).

NXFS is easily run by the user over an SSH tunnel. We provide a Python library function and command line tool to automate this step, and we have extended NeXpy to do so as well. Thus, NXFS runs as the Unix user on the file server.

V. Related work

Our system blends concepts from ad hoc metadata-rich systems (MCAT, MCS), remote filesystems (Chirp and SSHFS), and remote object systems (CORBA, Java RMI, etc.).

MCAT is the metadata service for the Storage Resource Broker (SRB) [20]. It stores and supports searches on system and user attributes associated with data collections in SRB. The Metadata Catalog Service (MCS) [21] maps attribute matches to logical files in a Replica Location Service (RLS) [22], which then maps to one or more replicas on real servers. The service is substantially generated from its API, using WSDL and SOAP, and provides a SQL query interface.

Chirp [23], [24] is a file system protocol with Unix-like semantics. It can easily be run in user space and allows users to share data by using a variety of authentication systems and rich ACL-based authorization. Chirp data can also be accessed by Parrot [25], which creates a virtual filesystem on which unmodified applications can run. Parrot does not require a kernel module; it uses the Linux ptrace interface to intercept system calls and replay them on remote filesystems. In addition to Chirp, Parrot supports FTP, HTTP, and other data services. Parrot has been used previously for high energy physics studies with data from the Large Hadron Collider (LHC) at CERN [26].

SSHFS [27] provides a Linux virtual filesystem over the SSH protocol. No server other than SSH [28] need be run. SSHFS is built on the FUSE [29] filesystem, a kernel module that allows the easier development of filesystem clients. FUSE modules are triggered by the kernel but run in user space, allowing user code to implement system calls. Given FUSE, the `sshfs` program sets up the SSH connection to the given server and establishes the virtual filesystem at the given mount point on the client.

CORBA [30] provides a language, OS, and hardware-independent object service. Originally designed for C, CORBA was extended to support C++, Java, and eventually many other languages. Its focus on interfaces allows widely disparate systems to be integrated, such as integrating older applications with newer services. Java Remote Method Invocation

(RMI) [31] can be used in pure Java systems for remote method access. Essentially, Java interfaces are compiled into client and server stubs to link Java programs. RMI+CORBA systems are also possible: CORBA interfaces can be compiled into RMI systems, and Java interfaces can be compiled into CORBA interfaces.

Pyro [19] flexes the powerful introspective features of Python to *dynamically* generate remote interfaces. Use starts when the user instantiates a Pyro service on the server. Python objects may then be registered with this service with a given name. Python clients can then connect to the Pyro service port and look up the desired object, obtaining an instance. Method invocation is relayed back to the server. Pyro transfers data using multiple optional techniques with varying performance and security levels, including Python's pickle or marshal features, or external JSON or Serpent techniques.

VI. Performance results

We measure the performance of the complete system by considering its component parts in detail.

A. Catalog

To evaluate the scalability of the catalog, we emulate an increasing number of datasets and annotations. We use a test client that uses the catalog REST API to invoke remote operations on a separate catalog instance. We hosted the test client and catalog on separate Amazon Web Services (AWS) EC2 instances in the AWS east-1 region: the test client on a t2.small (1 vCPU, 2 GB memory, 8 GB disk) and the catalog on a m3.medium (1 vCPU, 3.75 GB memory, 8 GB disk). We executed all client requests sequentially over several weeks. Reported results are measured at the client application and include the overhead of REST calls and network latency.

Figure 6 shows the time taken, as the number of datasets increases from 0 to 150,000, to create a new dataset, add 10 text annotations to that dataset, retrieve an existing dataset at random, and retrieve the 10 annotations of that dataset. We see that creating new datasets takes on average 0.49 s (min: 0.38 s, max: 31.54 s, median: 0.45 s, std dev: 0.24 s); adding 10 annotations takes on average 0.56 s (min: 0.47 s, max: 26.80 s, median: 0.54 s, std dev: 0.17 s); retrieving a dataset takes on average 0.36 s (min: 0.27 s, max: 15.50 s, median: 0.35 s, std dev: 0.15 s); and retrieving 10 annotations for a dataset takes on average 0.36 s (min: 0.27 s, max: 31.47 s, median: 0.34 s, std dev: 0.19 s).

Creating and adding annotations are the more expensive operations because they involve database insertions. Creating a new dataset involves the creation of eight annotations to record dataset metadata (e.g., owner, creation data, and ACLs). Retrieving a dataset involves fetching seven standard annotations (id, owner, modified, created, modified by, read, write) and thus takes appropriately the same time as retrieving 10 user-supplied annotations. We would expect annotation creation time and annotation retrieval time to increase with the number of annotations. These results show no increase with the number of datasets, suggesting that the catalog can scale well beyond the 150,000 datasets considered in our experiments.

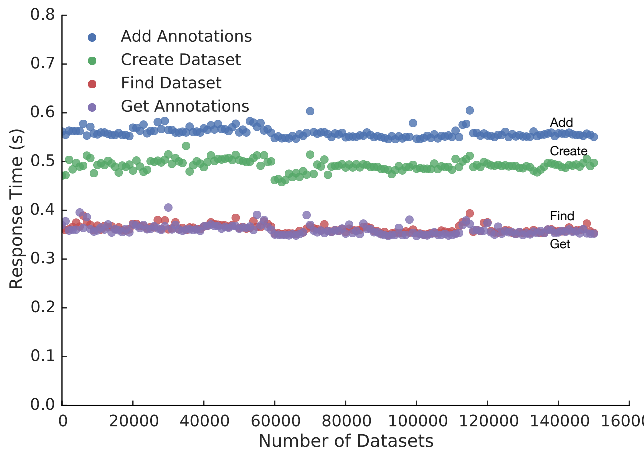


Figure 6: Catalog performance showing the time to create datasets, add 10 annotations to a dataset, query for a dataset based on an annotation, and retrieve a dataset's annotations. Data points are shown using a 1000 point moving average.

B. Server access rates

Once data has been located in the catalog, it can be retrieved from the data server. For these experiments we stored all bulk data on a computer with two Intel Xeon E5-2697 processors totaling 24 cores or 48 hardware threads, contains 400 GB RAM, and a 2108-based RAID controller for 24 disks. The server is not accessible to the Internet; it must be accessed through a login node via multi-hop SSH.

We access the data server from two sites, named **LAN** and **WAN**.

- **LAN**: We connect from a workstation onsite at ANL, connected end-to-end by a wired network. This case models the performance experienced by an APS user (i.e., on-site at ANL) accessing data during a run. The LAN client is an Intel i7-2760QM with 4 cores or 8 hardware threads, with 8 GB RAM.
- **WAN**: We connect from an EC2 instance. This case models the performance experienced by an APS scientist or visitor accessing the data from home, a hotel, or their home institution. The instance is an m3.medium, containing a Intel Xeon E5-2670 v2 and 3.75 GB RAM in availability zone us-east-1e (N. Virginia).

C. Raw performance

We next report on the raw network characteristics of the LAN and WAN sites.

1) Latency: We measured the basic network round-trip time (RTT) with a simple Tcl script that accessed `cat` on the data server over an SSH tunnel, averaged over 100 accesses. The RTT for LAN was 1.139 ms. The RTT for WAN was 22.95 ms.

2) Bandwidth: We measured the basic network bandwidth by SCP transfer times for varying file sizes, averaged over three attempts. Rates are shown in Figure 7. The LAN peaks at around 109 MB/s, and the WAN peaks at around 15 MB/s.

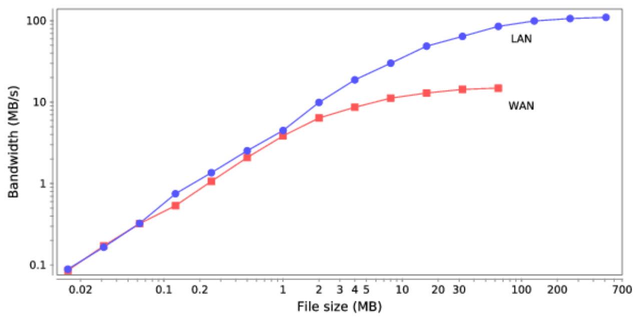


Figure 7: Raw bandwidth for LAN and WAN.

D. *NeXus file access*

We also measured the performance of different remote access mechanisms for NeXus data. We varied the underlying access method but not the test script (NeXus API calls in Python), as follows.

- **Local**: The test was performed on the data server; no data was transferred over the network. This control case measures the performance of the data server and its RAID filesystem.
- **SSHFS**: Data was served over SSH and presented to an unmodified NeXus/Python client accessing an SSHFS mount point.
- **Chirp**: Data was served by a Chirp server on the data server. An unmodified NeXus/Python client was run inside a Parrot environment, seamlessly accessing data on the data server.
- **Pyro**: Data was served by our NXFS system. The unmodified test accessed data behind the NeXus file object, implemented with our NXFS system.
- ***/L**: The given system was on the LAN.
- ***/W**: The given system was on the WAN.

We created a directory of 100 NeXus files on the data server. Each 14 GB file came from a real APS run and contained NeXus metadata in addition to a single 3D variable of size $1800 \times 2048 \times 2048$ (7.5B pixels), each a 32-bit floating point number (HDF type `H5T_IEEE_F32LE`).

The tests follow a typical user access of open, scan metadata, and access pixels.

1) File open rate: This test picked one file at random and opened it with the given mechanism. The reported number (Figure 8) is an average of 60 trials.

The Local file open time averages 50 ms but is noisy, as expected on our complex RAID filesystem. File opens with NXFS are competitive with Local performance, which is not surprising with 1 ms network latency.

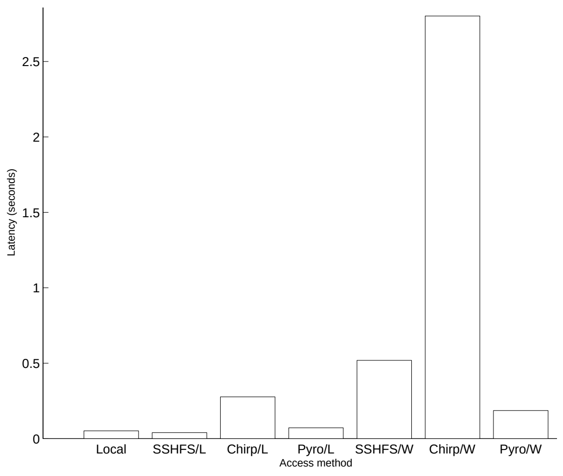


Figure 8: File open latency.

2) HDF browsing rate: The next step is to browse the HDF metadata of the given file with the NeXus API. This test picked one file at random and browsed down three levels of the hierarchy. The reported number (Figure 9) is an average of 60 trials and measures each lookup.

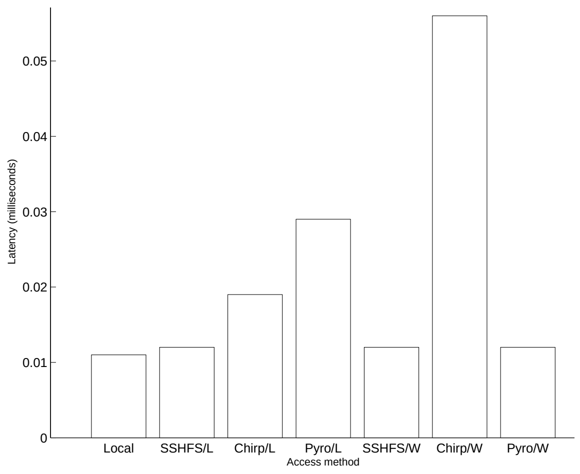


Figure 9: Metadata access latency.

All lookup methods run in at most tens of microseconds, indicating that the results are cached from open time, as expected. (Times are so small and noisy that results should not be compared with each other.)

3) Random access rate: The last step is to access pixels. In this case, we focus on latency, since plots over small fractions of the dataset are critical to APS users during experiments. (This test provides an estimate for small slab accesses as well.) A file is opened at random, and then either 100 (for LAN) or 10 (for WAN) random, noncontiguous pixels are retrieved. The reported number (Figure 10) is an average of three such trials.

For this test, the NXFS latencies are proportionate to Local access times, even for WAN. At 0.08 s, they are under the

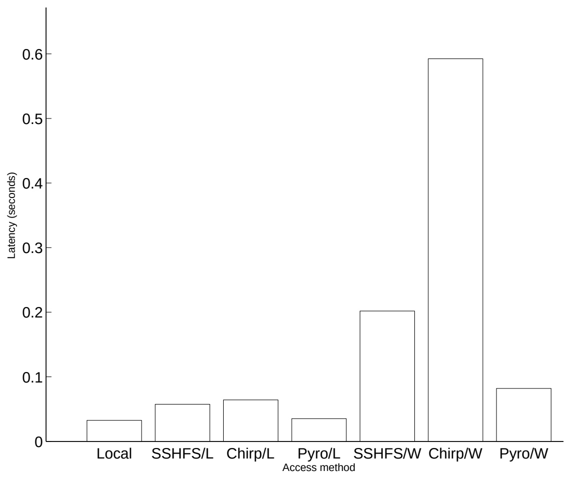


Figure 10: Random access latency.

effective frame rate of the human eye [32] and exceed the performance of the application-agnostic systems.

VII. Summary

We have presented a comprehensive system for remote access to data lakes produced by light source experiments. The system is highly modular and consists of a metadatabase (Globus Catalog §III), bulk data movement system (Globus Transfer §III-B), and remote object interface for interactive operation (NXFS §IV-B). This paper is the first presentation of the Catalog and NXFS systems and offers insight into the data management methods used at the APS. It also illustrates a complete, highly adaptable solution to indexing and accessing scientific big data in place.

Globus Catalog offers a highly collaborative, user-friendly metadata annotation system usable in multiple ways, from the web to Python interfaces to command line tools. It integrates well with the Globus Transfer system and supports data annotation at acquisition time, interactive queries, and long-term data management.

NXFS offers a mix of filesystem and object service features, seamlessly enhancing a data visualization toolkit with remote data access techniques. It integrates well with Globus Catalog and exceeds the performance of application-agnostic remote filesystem techniques.

VIII. Future work

We plan to enhance these systems in multiple ways and utilize them in other application areas.

With Globus Catalog, we are working to 1) deploy a new web user interface to improve user experience; 2) support hybrid storage models (e.g. entity schema for dense data and decomposed storage for sparse data); 3) build out additional functionality in the command line interface tools to match the underlying Python API client and support myriad scripting integrations (e.g. with SPEC at the APS); 4) enhance provenance tracking and subsequent visualization; and 5) integrate with various workflows in the x-ray community at the APS. We will also investigate the potential to integrate secure Globus HTTP endpoint access to allow simplified data access through the catalog web interface.

As we scale the Globus Catalog service to meet demand, we will investigate best effort approaches for when and where to provision new catalog service nodes. In addition, we will investigate best effort approaches for where to place new catalogs, in existing (or new) multitenant catalog service nodes. The multitenancy of the catalog services and its multiple nodes will require investigation of the security vulnerabilities of the system.

We are integrating new features into NXFS, including data modification operations and remote computation. This will allow users to visualize the results of data transformations, such as background subtraction or data projections, while moving only minimal amounts of data over the network.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, under contract DE-AC02-06CH11357, and by NSF award ACI 1148443. Computing resources were provided in part by NIH through the Computation Institute and the Biological Sciences Division of the University of Chicago and Argonne National Laboratory, under grant S10 RR029030-01, and by NSF award ACI 1238993. X-ray scattering experiments were supported by the U.S. DOE, Office of Science, Materials Sciences and Engineering Division, using the resources of the Advanced Photon Source, a U.S. DOE Office of Science User Facility, and the Argonne Leadership Computing Facility, both operated for the DOE Office of Science by Argonne National Laboratory under Contract No. DE-AC02-06CH11357. The authors acknowledge Ketan Maheshwari for support on EC2.

References

[1] Thomas Proffen and Reinhard Neder. DISCUS: A program for diffuse scattering and defect-structure simulation. *Journal of Applied Crystallography*, 30(2):171–175, 1997.

[2] Wiktionary. Definition of data lake.

[3] Margaret Rouse. What is a data lake? http://searchaws.techtarget.com/definition/data-lake.

[4] Ian Foster, Rachana Ananthakrishnan, Ben Blaiszik, Kyle Chard, Ray Osborn, Steven Tuecke, Mike Wilde, and Justin Wozniak. Networking materials data: Accelerating discovery at an experimental facility. In Gerhard Joubert and Lucio Grandinetti, editors, *Big Data and High Performance Computing*. In press, 2015.

[5] Takeshi Egami and Simon J. L. Billinge. *Underneath the Bragg Peaks: Structural Analysis of Complex Materials*, page 118. Elsevier, 2003.

[6] Mark Könnecke, Frederick A Akeroyd, Herbert J Bernstein, Aaron S Brewster, Stuart I Campbell, Björn Clausen, Stephen Cottrell, Jens Uwe Hoffmann, Pete R Jemian, David Männicke, Raymond Osborn, Peter F Peterson, Tobias Richter, Jiro Suzuki, Benjamin Watts, Eugen Wintersberger, and Joachim Wuttke. The NeXus data format. *Journal of Applied Crystallography*, 48(1):301–305, February 2015.

[7] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[8] NeXpy: A Python GUI to analyze NeXus data. http://nexpy.github.io/nexpy.

[9] Laura Wolf. Boosting beamline performance, 2014. https://www.alcf.anl.gov/articles/boosting-beamline-performance.

[10] Justin M. Wozniak, Timothy G. Armstrong, Daniel S. Katz, Michael Wilde, and Ian T. Foster. Toward computational experiment management via multi-language applications. In *DOE Workshop on Software Productivity for eXtreme scale Science (SWP4XS)*, 2014.

[11] Ian Foster. Globus Online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*, 15(3):70–73, May 2011.

[12] William Allcock, John Bresnahan, Raj Kettimuthu, Mike Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus striped GridFTP framework and server. In *SC'2005*, 2005.

[13] Kyle Chard, Steven Tuecke, and Ian Foster. Efficient and secure transfer, synchronization, and sharing of big data. *Cloud Computing, IEEE*, 1(3):46–55, Sept. 2014.

[14] Robert Schuler, Carl Kesselman, and Karl Czajkowski. An asset management approach to continuous integration of heterogeneous biomedical data. In Helena Galhardas and Erhard Rahm, editors, *Data Integration in the Life Sciences*, volume 8574 of *Lecture Notes in Computer Science*, pages 1–15. Springer International Publishing, 2014.

[15] Daniel Tunkelang. Faceted search. *Synthesis lectures on information concepts, retrieval, and services*, 1(1):1–80, 2009.

[16] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[17] Justin M. Wozniak, Timothy G. Armstrong, Michael Wilde, Daniel S. Katz, Ewing Lusk, and Ian T. Foster. Swift/T: Scalable data flow programming for distributed-memory task-parallel applications. In *Proc. CCGrid*, 2013.

[18] Justin M. Wozniak, Hemant Sharma, Timothy G. Armstrong, Michael Wilde, Jonathan D. Almer, and Ian Foster. Big data staging with MPI-IO for interactive X-ray science. In *Proc. Big Data Computing*, 2014.

[19] Irmen de Jong. Pyro web site. https://pythonhosted.org/Pyro.

[20] Arcot Rajasekar, Michael Wan, Reagan Moore, Wayne Schroeder, George Kremenek, Arun Jagatheesan, Charles Cowart, Bing Zhu, Sheau-Yen Chen, and Roman Olschanowsky. Storage Resource Broker - Managing distributed data in a grid. *Computer Society of India Journal*, 33(4):42–54, 2003.

[21] Gurmeet Singh, Shishir Bharati, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, and Laura Pearlman. A metadata catalog service for data intensive applications. In *Proc. Supercomputing*, 2003.

[22] Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, and Robert Schwartzkopf. Performance and scalability of a replica location service. In *Proc. High Performance Distributed Computing*, 2004.

[23] Douglas Thain, Sander Klous, Justin Wozniak, Paul Brenner, Aaron Striegel, and Jesus Izaguirre. Separating abstractions from resources in a tactical storage system. In *Proc. Supercomputing*, 2005.

[24] Douglas Thain, Michael Albrecht, Hoang Bui, Peter Bui, Rory Carmichael, Scott Emrich, and Patrick Flynn. *Data Intensive Distributed Computing: Challenges and Solutions for Large Scale Information Management*, chapter Data intensive computing with clustered Chirp servers, pages 140–154. IGI, 2012.

[25] Douglas Thain and Miron Livny. Parrot: Transparent user-level middleware for data-intensive computing. In *Proc. Workshop on Adaptive Grid Middleware*, September 2003.

[26] Dillon Skeehan, Paul Brenner, Ben Tovar, Douglas Thain, Nil Valls, Anna Woodard, Matthew Wolf, Tessa Pearson, Sean Lynch, and Kevin Lannon. Opportunistic high energy physics computing in user space with Parrot. In *Proc. CCGrid*, 2014.

[27] Miklos Szeredi. SSHFS web site. http://fuse.sourceforge.net/sshfs.html.

[28] OpenSSH web site. http://www.openssh.com.

[29] FUSE: Filesystem in userspace. http://fuse.sourceforge.net.

[30] Michi Henning. The rise and fall of CORBA. *ACM Queue*, 4(5), 2006.

[31] Qusay H. Mahmoud. Distributed Java programming with RMI and CORBA. http://www.oracle.com/technetwork/articles/javase/rmi-corba-136641.html, 2002.

[32] Paul Read and Mark-Paul Meyer. *Restoration of motion picture film*, pages 24–26. Butterworth-Heinemann, 2000. Cited in Wikipedia.