FILESTORE: A FILE MANAGEMENT TOOL FOR NSLS-II BEAMLINES

A. Arkilic, L. Dalesio, D. Chabot, W.K. Lewis, D. Allan, T. A. Caswell
Brookhaven National Laboratory, NSLSII, Upton, NY, USA

Abstract

NSLS-II beamlines can generate 72,000 data sets per day resulting in over 2 M data sets in one year. The large amount of data files generated by our beamlines poses a massive file management challenge. In response to this challenge, we have developed filestore, as means to provide users with an interface to stored data. By leveraging features of Python and MongoDB, filestore can store information regarding the location of a file, access and open the file, retrieve a given piece of data in that file, and provide users with a token, a unique identifier allowing them to retrieve each piece of data. filestore does not interfere with the file source or the storage method and supports any file format, making data within files available for NSLS-II data analysis environment.

INTRODUCTION

Current state-of-the-art detectors are capable of producing megapixel images with frame-rates in the kilohertz range which may result in peak data rates of up to 800 Mb.s-1. As a result, the volume of data associated with an experiment may now exceed several tens of terabytes so simply moving data from one location to another is prohibitive. As such, it may not be possible for external users to take a copy of all the data on portable media or to easily transfer over the Internet. Thus it requires high-performance data storage systems and advanced processing capabilities to allow users to access their data both during their experiment at NSLS-II and to continue the analysis following the NSLS-II visit. The middle-layer architecture of NSLS-II beamlines provides such capabilities using a blend of NoSQL databases and distributed file systems [2]. **fileStore** is the component of the middle-layer that keeps track of experimental files. These experimental files are generated via EPICS areaDetector module or any custom camera. filestore's NoSQL mongo backend allows users to save any sort of information about their images, while providing links to metadatastore runs[1]. Filestore provides clients with high-performance data mining techniques that were not possible via legacy applications in place.

ARCHITECTURE

filestore provides information regarding the location of the file within the General Parallel File System (GPFS). MongoDB serves as the back-end to filestore due to its flexibility and performance. Just like metadatastore, filestore has a set of required standard fields and allows users/data acquisition scripts to add any field in any hierarchy, making it a primary storage environment for image headers [1].

```
In [1]: from filestore.api import insert_resource, insert_datum

        resource_id = insert_resource('csv', 'example.csv')
```

Figure 1: filestore Python API Resource Insert Example.

Filestore's capabilities far exceed providing URL information. It also serves as a platform for performing data file formatting (via handlers) and data regression (via its complex querying capabilities). Experimental files can be stored completely independent of filestore's acknowledgement. Since filestore does not need to be aware of the file at time of creation, it is possible to migrate data files from legacy systems into the modern middle-layer service framework of NSLS-II beamlines (see Fig. 1).

```
In [5]: insert_datum(resource_id, 'some_id1', {'line_no': 1})
Out[5]: <Datum: Datum object>

In [6]: insert_datum(resource_id, 'some_id2', {'line_no': 2})
Out[6]: <Datum: Datum object>

In [7]: insert_datum(resource_id, 'some_id3', {'line_no': 3})
Out[7]: <Datum: Datum object>

In [8]: insert_datum(resource_id, 'some_id4', {'line_no': 4})
Out[8]: <Datum: Datum object>

In [9]: insert_datum(resource_id, 'some_id5', {'line_no': 5})
Out[9]: <Datum: Datum object>
```

```
In [10]: from filestore.api import register_handler

In [11]: register_handler('csv', CSVLineHandler)
```

Finally, we are ready to retrieve that data. All we need is the unique ID.

```
In [12]: from filestore.api import retrieve

In [13]: retrieve('some_id2')
Out[13]: 'b\n'
```

Figure 2: Datum and Handler example.

ISBN 978-3-95450-148-9

Write a Handler

```
In [14]: import h5py

In [15]: class HDF5DatasetHandler(object):
   ....:     def __init__(self, filename):
   ....:         self.file = h5py.File(filename)
   ....:     def __call__(self, key):
   ....:         return self.file[key].value
   ....:
```

Make a Record of the Data

```
In [16]: from filestore.api import insert_resource, insert_datum

In [17]: resource_id = insert_resource('hdf5-by-dataset', 'example.h5')

In [18]: insert_datum(resource_id, 'some_id10', {'key': 'A'})
Out[18]: <Datum: Datum object>

In [19]: insert_datum(resource_id, 'some_id11', {'key': 'B'})
Out[19]: <Datum: Datum object>

In [20]: insert_datum(resource_id, 'some_id12', {'key': 'C'})
Out[20]: <Datum: Datum object>
```

Retrieve the Data

```
In [21]: from filestore.api import register_handler, retrieve

In [22]: register_handler('hdf5-by-dataset', HDF5DatasetHandler)

In [23]: retrieve('some_id11')
Out[23]:
array([[[1, 9, 4, 2, 4],
        [0, 1, 1, 8, 6],
        [6, 8, 3, 0, 2],
        [7, 3, 7, 8, 5],
        [1, 2, 8, 8, 2]]])
```

Figure 3: A sample HDF5 file handler, save, and retrieve file example.

In NSLS-II experiments, experiment session manager library, **BlueSky**, notifies filestore regarding the arrival of the experimental data using filestore's Python client API. In order to save/retrieve image file(s), filestore requires two pieces of information: how to access and open the file (or, generically, "resource") and how to retrieve given pieces of data in that file (Fig. 2). Just like metadatastore, filestore provides a token, a unique identifier, which clients can use to retrieve each piece of data. This way, given an **event** in metadatastore, clients can access files generated during that specific event. **File Handlers** aim to eliminate differences in N-dimensional data representation by converting files with any extension to a **numpy** array (Fig 3). This allows filestore to support various beamlines as it provides support for any file format given clients provide the file handlers. It also makes it possible to develop data analysis tools that expect a standard input for N-dimensional data.

FUTURE

We are aiming to use EPICS v4 in order to stream the N-dimensional arrays in NSLS-II experiment files to clients. As seen in Fig. 4, specific v4 normative type, NTUnion, allows filestore to ship complex data structures that include the MongoDB documents that contain image header information and images themselves in rates that are much higher than traditional EPICS Channel Access Protocol.

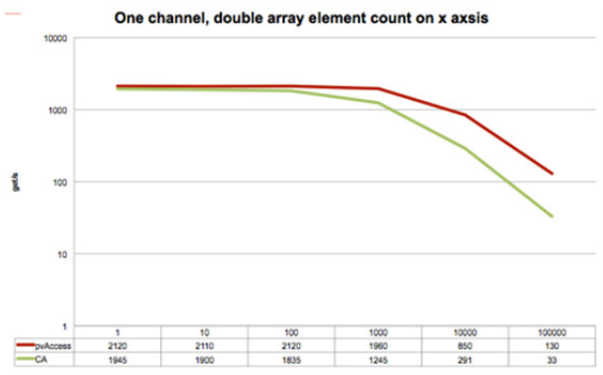

Figure 4: N-dimensional Array Stream CA vs pvAccess.

CONCLUSION

Filestore reduces the latency of accessing and moving of large data sets from minutes to seconds. Its modularity and ability to provide links to metadatastore events, allow NSLS-II data analysis environment to perform complex data regression operations. File Handlers makes it possible to write data analysis tools that expect single standard data format, numpy arrays.

ISBN 978-3-95450-148-9

REFERENCES

[1] metadatastore: A primary data store for NSLS-II beamlines, A. Arkilic, L. Dalesio, D. Allan, W. K. Lewis, Presented at the ICAPLEPCS 2015.

[2] NSLS-II High Level Application Infrastructure and Client API Design, G. Shen, L. Yang, K. Shroff Presented at the2011 Particle Accelerator Conference.

ISBN 978-3-95450-148-9