DUNE Data Model Meeting: Metadata

Metadata Needs And Considerations

Steven Timm

The following are design concerns for whatever new metadata system is written or adopted. They are written from the point of view of data management operations and the features we believe are needed to have a system which is serviceable and operable. I am purposely trying to avoid issues of implementation or format and focus on the purpose for the system.

Purpose of the Metadata System

The metadata system is meant to tell us: (1) What type of data is this file. (2) How was it made, what hardware and software versions (3) When was it made (4) If it is a derivative file, from which files was it derived, and (5) is this file meant to be persistent or is it temporary.

The metadata system is *not* meant to tell us: What was the voltage of wire # 2 in APA # 6. What was the phase of the moon on the day of data taking, which events are in this file. It may however contain pointers to the database systems where that information is available.

Current State

We are currently using a hybrid of SAM and Rucio for metadata and file transfer. At the moment all Rucio locations of data that we have, are still declared as file locations in the SAM database and interactive and online users still query SAM for locations of the files and use SAM utilities to fetch the files to their job.

The next intermediate step would be to have all SAM tools just do pass-through to rucio commands to determine file locations and to fetch files. We would then be in a situation where information about the nature of each file would be stored in SAM metadata but the location would be stored in Rucio.

Current understanding of project goals

The DUNE Data Management Project as currently planned has three major activities: (1) transitioning to have all file location information only in Rucio, (2) to have a new metadata server to replace the SAM metadata service, and (3) to make new clients to be able to use this information. There was a Rucio file metadata server written at some point but it has never been tested at scale nor has it been deployed by any of the big experiments currently using Rucio. It is the opinion of FNAL developers and also of the core Rucio team that we are better to develop this outside of the main Rucio framework.

Facility goals:

We have strong encouragement from Fermilab and the other tape resource providers (CERN, RAL, CC-IN2P3) to build data life cycle into our data planning from the start. DUNE has existing data retention policies (linked in the master data model workshop document) but further technology is needed to implement them.

Data Retention Policy:

DUNE data retention policy currently is DUNE DOCDB 5752
How do we indicate the anticipated retention policy in the metadata? Rucio has the notion of location rules that expire after a certain amount of time, but not of files that expire. Rucio can support several different methods of file management including deleting junk only when space is needed to be reclaimed. Can we encode the expiration time in the metadata, or should we push for a Rucio feature to do this? Should we consider having a dead-man-switch model in which if a file is not explicitly and actively renewed by the owner it goes away? Likewise should we have some tag that pins the file in Rucio forever? We are designing a system that will live beyond the end of the 32-bit Unix epoch.

File Families:

Each tape-backed storage has the notion of file families. They are all implemented slightly differently. DUNE would do well to define our own internal notion of tape file family and then implement in the rucio site-based wrappers how we interface with the file families as defined in Enstore, Castor or other tape systems to come. File family should be tagged in the metadata and in rucio itself if possible. File family creation must be automated.   Also need to address how do we specify quality of service--do we need fast disk, slow disk, lots of reads, few reads, etc.

Consistency:

If there are two systems which have information about a file (Rucio and the metadata catalog) then some care will be required to make sure they remain consistent, namely that every file in Rucio has a metadata entry and vice versa. If a file is deleted from rucio should the metadata entry be deleted too? (In SAM it was). Also how do we set up procedures such that the files that Rucio says are on the SE, actually are on the SE, and no files unknown to Rucio are on the SE. If the Art framework or its successors are involved, then there is also metadata stored within the data file itself, this provides a third place where conflicting information can be stored and we will have to deal with that.

Hierarchy:

There are metadata variables that properly apply to all files in a run or a subrun. In the SAM methodology every file has standalone metadata. In Protodune we found this to have two limitations. One--there were some flags (good run / bad run) which were not known until the run was over and many of the files had already been declared to SAM, and yet they needed to be in the metadata. Two--there was by nature a lot of duplication of a large number of fields that were common to runs, and thus a lot of data duplicated in the database. We should discuss if it is possible to assign metadata attributes to all files in a run, all files in a subrun, or all files in a rucio “data set” or “container.” This may be desirable but it may come at the cost of too complicated a schema.

Philosophy:

Garbage bin or Fort Knox? What authentication needed to write or read? What validation if any should be done? Do we allow tuples, lists, etc, or just scalar values? Are certain tags only allowed to take an enumerated set of values? (Since we have to incorporate a lot of SAM metadata the answer probably has to be YES on all of the above). Do we allow pointers and links to external databases and tables? Metadata has historically been declared in JSON format, which many programming languages can already easily manipulate. Do we intend to let users declare metadata into this system (and/or do we let them declare files to rucio)? If users do, then how do we clean up data from defunct users?

Interfaces:

Database backend: Recent NOvA experience shows that the database backend is important--both in software design and hardware design, and that the metadata database is both complicated and large. We will have to include database group and DBA’s in the discussion from the beginning.

DAQ frontend: For the first protodune-SP run a significant amount of development was done of plugins that ran inside ARTDAQ to make sure we got the metadata we needed, by Jeremy Hewes of Cincinnati. We will need to make sure that there is a person of similar skill embedded in the DAQ group to make sure that the metadata is available for ingest.

DAQ consumers: In early days of ProtoDUNE there was (and still is) high pressure to know which event was in which file. We accomplished this in ProtoDUNE by jamming the whole event list into a table and linking it to the metadata. This is not a scalable plan going forward. Rather data management should be making a data fetcher/server that can deliver event X of run Y. (Or even just a single sub block of event X).   This will be vital not only for detector commissioning but for analysis going forward

Service Level

What are uptime requirements? Does DUNE DAQ need to be in contact with metadata server at all times? Should database be distributed, cache, otherwise?

Do other Open-Source Solutions exist?

“Dublin” core --are there others? Other experiments?