MetaCat - metadata catalog for data management systems

Mandrichenko Igor[1]

[1]FNAL, Scientific Computing Division, Batavia, IL, USA

Abstract. Metadata management is one of three major areas of scientific data management along with replica management and workflow management. Metadata is the information describing the data stored in a data item, a file or an object. It includes the data item provenance, recording conditions, format and other attributes. MetaCat is a metadata management database designed and developed for High Energy Physics experiments. As a component of a data management system, it's main objectives are to provide efficient metadata storage and management and fast data selection functionality. MetaCat is required to work on the scale of 100 million files (or objects) and beyond. The article will discuss the functionality of MetaCat and technological solutions used to implement the product.

1 Metadata catalog as part of a data management system

A data management system built for HEP experiments is responsible for storing, moving and making available physical data collected by the collaboration. Typically, the data is organized as a collection of objects or files. If it is a file, then usually it contains a set of records of observed HEP events. When the data management system deals with objects, they can be either something like a reconstructed event or a collection of events. For the purpose of this paper, it is not important whether the data management system deals with files or objects. Therefore, we will be mostly using the term "file", but it can be replaced with the term "object" without making any changes to the rest of the paper.

Having said that, we can say that a data management system operates on a collection of "physical" copies of files (or objects), moving them between storage elements and making them available to the data processing and analysis. Typical HEP data management system can be broken into 3 major components or areas of functionality, as shown in Figure 1:

- Replica Manager - the part of the system responsible for moving data between storage elements. Replica Management system usually operates in terms of replicas - copies or "physical" representations of "logical" files. Single "logical" file can

1 Corresponding author: ivm@fnal.gov

have multiple "physical" replicas in multiple storage elements. Replica Manager keeps track of all physical replicas of logical files and coordinates movement of the replicas.

- Workflow Manager - the component, which coordinates data consumption by data processing or analysis, making sure that all selected logical files get processed or analyzed as their replicas become available to the processing or analysis processes.
- Metadata Catalog, which stores metadata about logical files and makes it possible to select "interesting" logical files based on criteria expressed in terms of file metadata.

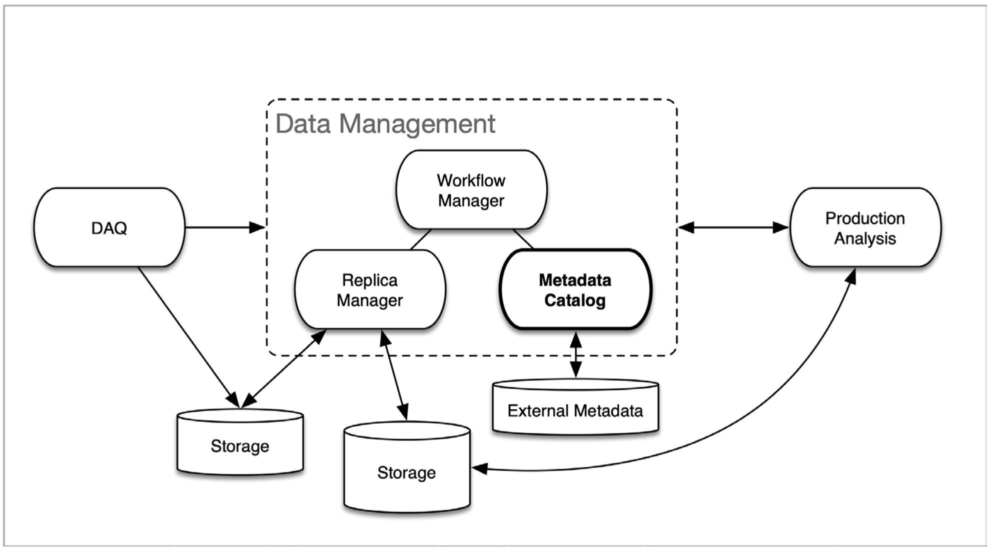

Figure 1. Data management structure

2 MetaCat

2.1 Project objective, scope and requirements

MetaCat is a project with the objective to build a metadata catalogue, which can be used in a HEP data management system. Although the main target user of the project happens to be ProtoDUNE/DUNE, MetaCat is required to be generic enough to be used by multiple experiments without any changes. It means that MetaCat has to satisfy the following general requirements:

- Metadata representation must be powerful, flexible and abstract enough to accommodate a wide range of metadata data types and possibly complex metadata structures
- The ability to scale to 100 million files and beyond
- The unit of operation of the catalog is an abstract "file" or "object" with minimal set of predefined attributes and the ability to add user-defined attributes in a flexible and convenient way
- File selection mechanism must be powerful and at the same time simple enough to express sufficiently wide range of metadata selection criteria
- MetaCat must be compatible with Rucio [1], which is a popular replica management system used by many HEP experiments, including DUNE/ProtoDUNE. On the other hand, it should be generic enough to work with other replica management systems.

Project scope was defined as building a metadata database with convenient user and application programming interfaces so that it can be used by humans as well as other components of the data management system. MetaCat *does not* have any knowledge of physical file replicas.

Based on our experience with SAM [2,3], an important requirement for MetaCat is to provide a mechanism to use data from external metadata sources, such as conditions or runs databases, as part of metadata queries without copying the external data into or making it appear as part of the metadata database.

2.2 Data model

MetaCat data model is illustrated in Figure 2.

MetaCat objects are identified by *names* within *namespaces*. The name of an object is unique within its namespace. A namespace/name pair uniquely identifies an object in MetaCat. In addition to namespace/name, files are assigned unique text *identifiers*. These identifiers are primarily for internal use within the MetaCat database, but are available to the user. At the time of the file declaration, the user can specify a unique file identifier, otherwise MetaCat will generate one. After a file is declared to MetaCat, it can be renamed. Both namespace and name can be changed, providing the new namespace/name pair is unique and the user owns the new and old namespaces. File identifier can not be changed.

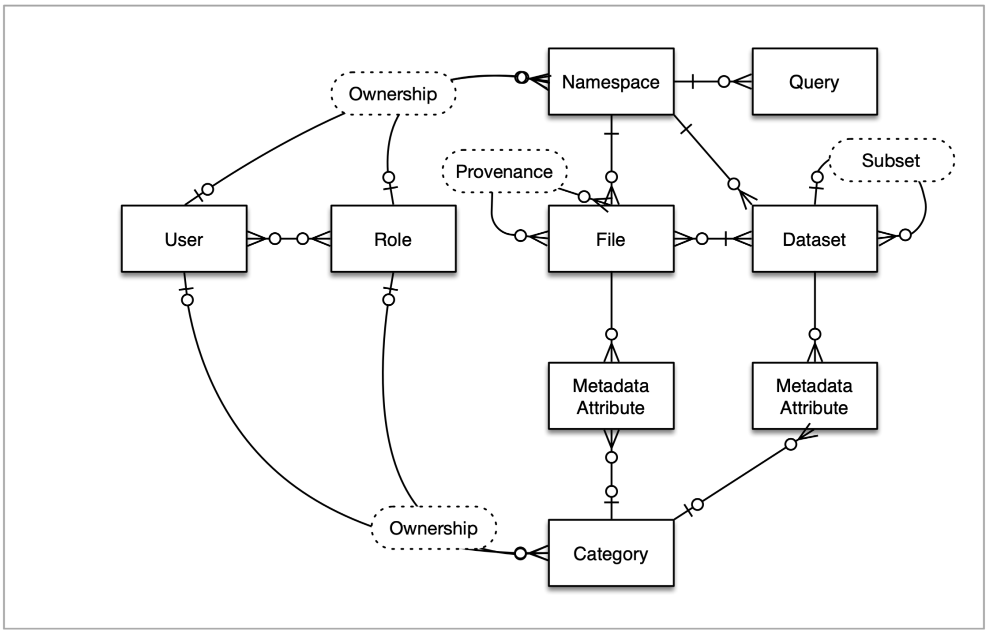

Figure 2. MetaCat data model

MetaCat stores metadata associated with files and datasets.

A *dataset* is a collection of files. A dataset can have zero or more *child datasets*. The system makes sure there are no circles in the parent/child relationship between datasets. A dataset can be unrestricted or set to be *monotonic*, which means files can not be removed from it or *frozen* - files cannot be added or removed from the dataset.

Files are added to and removed from a dataset explicitly. A file can belong to zero or more datasets. There is no requirement that file namespace has to be related in any way to the namespace(s) of the dataset(s) the file belongs to. If a file belongs to a child dataset, it *does not* mean it belongs to its parent dataset. However, the query language discussed below allows selection of files from child datasets, recursively.

There is a many-to-many provenance relationship between files. A file can have zero or more derived (*child*) files and zero or more *parent* files. The system makes sure the provenance relationship is not circular.

Metadata attributes have names and values. Any file or dataset can have zero or more metadata attributes. Attribute names are alphanumeric words optionally combined with dots. *Any* JSON structure can be an attribute value. Therefore, a file or a dataset attribute set is a JSON dictionary.

2.3 Ownership and permissions

A MetaCat user in the system is identified by a unique username. A user can have zero or more *roles*. Namespaces and attribute categories have owners. Namespace or category owner can be either an individual user or a role. If the namespace or the category is owned by the role, it means it is automatically owned by all role members.

The namespace owner (either directly or via the role membership) automatically owns all the datasets and files in this namespace. Ownership does not automatically propagate along the dataset parent/child or file provenance relationships.

Only the dataset and file owner can add the file to or remove the files from the dataset. Only the owner of the file or the dataset can change their metadata attributes.

2.4 Queries

One of the most important parts of the MetaCat functionality is the ability to query the database for “interesting” files. Essentially, the query is a logical expression in terms of file and/or dataset metadata attributes, file provenance relationship and dataset parent/child relationship. There are two types of query in MetaCat - *file queries* and *dataset queries*. A file query returns a set of files whereas dataset query returns a set of datasets. The returned sets of files and datasets lists are not guaranteed to be sorted in any particular order.

A query is merely a formula specifying the selection criteria. MetaCat never saves the results of the query, so re-running a query can produce different results as the files are added/removed from the system, to/from the datasets or their metadata attributes change. However, there is an option to save results of the file query as a new dataset or add selected files to an existing dataset.

A query can be saved into the database under a name within a namespace. This function can be used to publish complicated queries and make them reusable by other users.

2.5 MetaCat query language (MQL)

A MetaCat user writes the query in the query language called MQL. MQL allows the user to specify file/dataset metadata attribute criteria, use dataset parent/child relationship and file provenance relationship to select a set of file or a datasets. Further, simple queries can be combined into more complicated ones using logical operations like union, intersection (join) and exclusion. Named queries can be referred to from within the MQL expression by their name.

The most basic MQL query looks like this:

```
files from prod:dataset_A
```

This query returns all the files in the dataset "dataset_A" in namespace "prod". "files" and "from" are reserved words in MQL.

A more complicated query could look like this:

```
files from prod:dataset_A
 where daq.run_number < 1000
and format.type = 'root'
```

This query returns all files from prod:dataset_A, matching some criteria. "daq.run_number" and "format.type" are attribute names and these attributes are used to select certain files from the dataset.

Queries can be combined into other queries using set operations:

```
union(
 files from prod:dataset_A
       where daq.run_number < 1000,
 files from prod:dataset_B
       where daq.run_number > 1200
) where format.type = 'root'
```

This query combines two simple queries and then applies additional metadata criteria to the results.

A query can combine metadata criteria for files and datasets:

```
files from users:'data_2020_%'
 having type='mc'
 where format.type='hdf5'
```

Here, the "having" keyword indicates the dataset metadata criteria and "where" condition applies to files. '%' is the wildcard character for dataset names.

2.6 External data sources

MetaCat has the functionality to access external metadata sources like conditions databases and use the metadata stored there to filter file selection results. In order to make an external metadata source available to MetaCat instance, the collaboration develops a Python module with a standard interface, which knows how to access the data source and plug the module into MetaCat instance. Once the module is plugged into the instance, it can be referred to in the MQL query as a named *filter* by and used to filter results of an intermediate query or queries within the MQL expression or even "inject" metadata from the external source into the query and make it available for the selection criteria.

In MQL, access to the external data source is done via the "filter", referring to the plugged module. For example, the following fragment uses the module named "run_quality_limit":

```
filter run_quality_limit(50)
(
 files from users:'data_2020_%'
       having type='data'
)
```

This query could be used to select files from multiple datasets with run quality greater than 50, while the data quality is in fact stored in some external runs database. The filter would receive the results from the input query with all the metadata and then get the runs data from the external source and filter out files.

2.6 Architecture and Implementation

MetaCat is a typical web services database application. Its architecture is shown in Figure 3. The underlying database is not exposed to the users. It can be accessed via Python API, but the primary method of interacting with the system is through the web service or web GUI. The REST-based web service adds the scalability and cacheability to the system and completely unties the server side from the client implementation. Any standard HTTP/HTTPS client can interact with the system either directly or through a standard HTTP proxy or cache.

The system publishes the following interfaces:

- Direct database access Python API
- Web services REST interface
- Python client side API, which communicates with the server via HTTP
- Command line interface with basic set of commands to enter (modify) data into the database and to query the database

MetaCat is implemented in Python3 with PostgreSQL database. It uses PostgreSQL support of JSON data type to store file and dataset metadata and to perform efficient metadata queries.

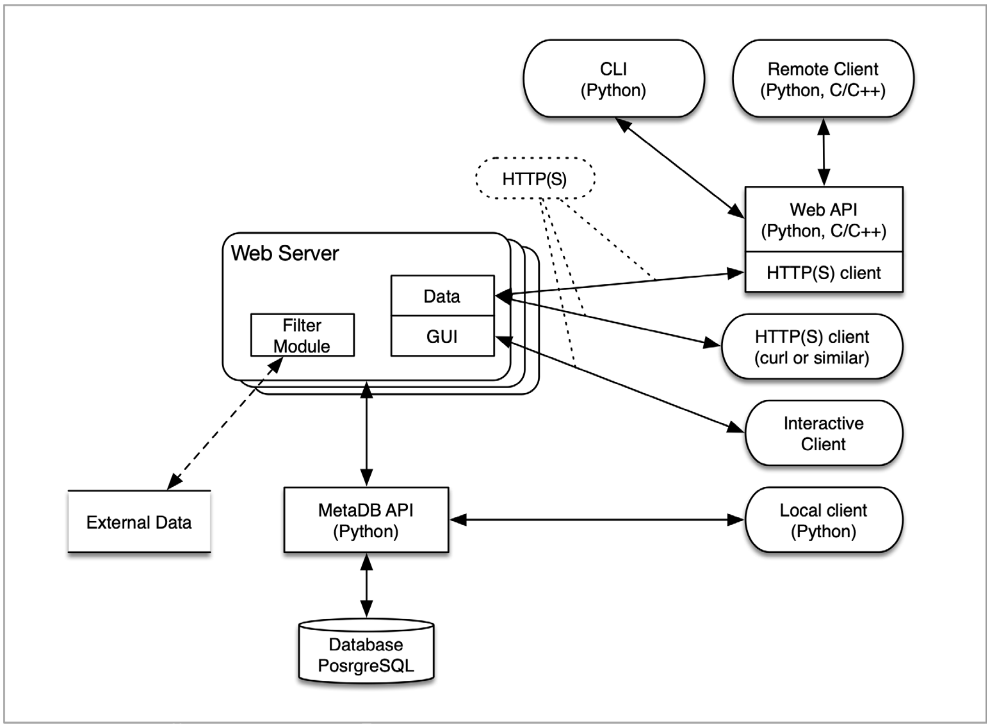

Figure 3 MetaCat architecture

3 Project status

The MetaCat project has been in development since about a year ago. Main target of the project happens to be DUNE/ProtoDUNE, but MetaCat is a general–purpose tool, designed to work in any data management system without any DUNE specifics in its design, implementation, interfaces or functionality. Currently MetaCat is in stable beta-testing stage, continuing its development based on the feedback received from users who are testing the product.

References

1. Rucio - https://rucio.cern.ch/
2. R. A. Illingworth 2014 *J. Phys.: Conf. Ser.* **513** 032045
3. S. Fuess *et al* 2017 *J. Phys.: Conf. Ser.* **898** 062036