Requirements and Implementation Ideas of a Metadata Catalogue

The ARDA Project

Editor: B. Koblitz

Abstract: We present a design proposal for a generic metadata catalogue which has an interface inspired by the POSIX standard. Experience with a small prototype implementation has been performed to validate this design. Streaming of results allows the server to operate with a small memory footprint and eases problems with timeouts.

Definitions:

Metadata is key-value pairs associated to a file where the key is a 0-terminated string and the value can be any binary data of a given size. This corresponds to the POSIX definition of file metadata, also termed extended file attributes, and allows to transparently store files with attached metadata into metadata aware file systems which are currently coming widely into use.

Metadata can be attached to directories, in this case files inherit the metadata as defaults. Here, directories are any collection of files defined in the file-catalogue. A directory does not need to have the semantics of a directory in a UNIX file system.

Copying a file copies also the metadata associated to the file.

In the case of the Grid, metadata will be associated with an LFN.

Rationale for this Design:

Using the POSIX definition of metadata for grid-metadata allows to copy files and their metadata to local file systems preserving the semantics of this metadata. Using this design allows also to use the well though through POSIX interface to this metadata.

Attaching metadata to the LFN distinguishes the metadata catalogue from the file catalogue where metadata-like access controls for files are associated with GUIDs. Associating the metadata with LFNs allows the user to make very flexible user of metadata: Every user can have his own metadata associated with a file or have different metadata associated with a file depending on the context (directory it is stored in), e.g. one directory may contain MC-files with their creation date and software versions as metadata in the context of the MC production system while a physicist analysing MC-files may have only parameters of the employed generators as metadata.

Attaching metadata to the LFN allows to group metadata by directory and thus provides a hierarchical structure allowing to distribute metadata over sites. Using directories to group files with similar metadata also allows to efficiently search files matching certain criteria in individual directories.

However, if a file catalogue is chosen without providing a directory structure (or in the case of files without LFN where the GUID can be used as index), it would still need to provide the functionality to group files in order to allow efficient searching.

A Possible Implementation:

Every directory containing files with metadata is represented by an SQL table with all keys as columns and all files in the directory as rows with the respective values. This allows fast searches for files in a directory. It also allows users to have different metadata schemas. Adding or removing metadata from a file may alter the columns of the table. Setting metadata for a directory changes the

defaults for the column values of the table (a variant may be a row with defaults for the directory).

Experience with existing metadata prototype implementations [AMI, REFDB] suggests to keep the protocol overhead due to a server in front of the database backed as small as possible by offering a minimum functionality and not encapsulating the responses of the database directly (e.g. by SOAP). To keep the memory footprint of the server even for very large responses, it is necessary to stream the response to the client. This is a key feature which most of the systems analysed by ARDA miss so far.

A possible implementation is to have a small authenticating server in front of the database taking encrypted plain text commands and returning a (streamed) text response, these text-strings can be then UU-Encoded and sent using SOAP as a transport layer. This scheme is currently implemented by the glite authenticating user interface. The front-end could also contain an indirection layer which would forward client request to a different database or refer the client to this other database. In the following a protocol for querying the metadata server is described as well as a C-API. The design of the API is inspired by POSIX, the protocol is a simple text-based protocol optimized for streaming.

Proposal for a command interface:

The following commands can be sent as text to the database front end. The response will be a line with a number as OK/Error code and then a list of strings, each string on a single line, this list of strings is terminated by the EOT character:

`getattr file key`: Returns the value of the key associated with the file as a text-string. If the internal storage type of the value is not ASCII, the data is converted to text.

`setattr file key value [type]`: Sets the value associated with the key of the file. Type is an SQL type. If type is omitted the default type in for this key in the directory is used, if there is no default type, the default is a text-string. The type only affects internal storage of the data and may be ignored by the implementation (e.g. if the backend is the filesystem).

`listattr file`: Returns all keys of a file text-strings. List is terminated by empty string.

`removeattr file key`: Removes an attribute corresponding to key.

`gettype file key`: Returns the internal storage type of the attribute as a text string describing an SQL standard type.

`find pattern query`: Returns a list of file names matching pattern and fulfilling the SQL query on their attributes. The list is terminated by an empty string.

`getattrx pattern key1 key2 ...`: This is a possible command which allows bulk queries for multiple keys of several files, saving DB queries. Returns a line with a filename matching the pattern and then the values of the keys line by line, for all matching filenames.

The commands can contain quoted strings e.g. the SQL query could be 'tracks > 10'. The responses must be plain ASCII. In any case the end of line and EOT characters need to be escaped. If it is necessary to store binary data, this can be done by storing UU-Encoded strings.

The key feature of this protocol is the possibility to stream it, allowing for a small memory footprint on the server. It can be in addition easily authenticated/encrypted using GSI.

Proposal for a C API:

The C-API described below is based on the POSIX API for metadata access of files. I uses the command interface to the metadata catalogue as described above to contact the metadata catalogue. It will issue the respective commands and interpret the results. The interface of the first four com-

mands of the API are pure POSIX and the definition can be studied in the man-pages located at http://acl.bestbits.at/man/man.shtml:

`getxattr()`: Read an attribute using a key

`setxattr()`: Set key value pair

`listxattr()`: Show all keys/values of a file

`removexattr()`: Remove an attribute by key from a file

Searching is done in directories (and in sub-directories if wanted) as an extension of POSIX, this command can be easily streamed, the interface is similar to opendir():

`md_handle *findFiles(const char *pattern, const char *statement):`

Find all files matching a pattern where a given SQL statement is true

The md_handle can be used to consecutively loop over the filenames via:

`int getfile(md_handle *handle, char filename[size], int size):`

Reads in the next filename and returns it (if filename is large enough).

`int free_md_handle(md_handle *handle):`

Frees the resources allocated by findFiles.

To read multiple attributes of several files matching a given pattern, the following command can be used:

`md_handle *getattrx(const char *pattern, char *keys[], int n_key):`

Returns handle to filenames matching pattern and values for keys[] of these files.

In addition to the getfile() (which reads in also the next set of values) and free_md_handle() calls above, a call to access the value corresponding to the nth key is necessary:

`int getvalue(md_handle *handle, int n, char *value, int v_size):`

Returns value in value of the nth key.

The construction using the `md_handle` structure allows streaming the entries for all files queried one by one to the client. It is similar to the ODBC C-Interface to SQL databases which also operates with handles to the data and allows for streaming.

Remarks:

It needs to be defined whether there is a size limit on the metadata, the size of this may be an implementation detail like in file systems, database systems typically have size limitations for their data types. Implementation of the DB schema itself is also an implementation detail. It would be possible to have schema evolution or support for fast addition or removal of columns.

To validate the C API, a prototype was built. The database backend is PostgreSQL which is accessed by a multi-threaded server (implemented in C++) using the ODBC abstraction layer. A C++ client library talks to the Server using the described text protocol using a TCP/IP connection. This library has an interface which resembles the C API described above. C++ clients built on this library allowed to explore and test the following issues:

- Usability of the API: The API seems to be minimal, rather complete (if many updates are expected a bulk-update command could be added) and performant.
- The memory footprint is small (and independent of the query-size!): 128KB per connection for

the server thread plus a database instance per connection (ODBC allows connection pooling which would make it possible to serve all server threads with only one database instance).

- The test implementation is stable and scales well (60 concurrent connections were reached transferring a total of 1.2GB of data without a noticeable performance penalty on a desktop computer).
- Having metadata- and file-catalogues implemented in one database allows an effective system for access control based on ACLs for metadata without noticeable performance loss.

At this point we are confident that the proposed interface is a reasonable starting point for a File-Metadata Catalogue which can be used as a foundation to implement the file-metadata or other metadata catalogues for the HEP experiments.

Links:

[AMI] http://lcg.web.cern.ch/LCG/peb/arda/public_docs/CaseStudies/ami_new.pdf
[REFDB] http://lcg.web.cern.ch/LCG/peb/arda/public_docs/CaseStudies/refdb_draft_v0.2.pdf