Overview of IDS Internals

Rolf Krahl

October 8, 2018

1 Introduction

The ICAT Data Service (IDS) operates with delayed actions, internal states, and multiple threads. This makes it somewhat difficult to understand from mere reading the sources what actions may be performed under certain conditions. The knowledge of the internal processes in IDS may be needed to properly implement a storage plugin in a non-trivial setup. In particular, one might need to know the context that each of the plugin's methods may be called in. This text shall provide a reference to accommodate this need. It is however only an overview. In order to keep the representation clear, many details have been omitted. Furthermore it is somewhat biased towards the point of view of a storage plugin developer and it is restricted to the case that the IDS is configured as a two level storage with the storage unit set to dataset. The text is based on IDS server 1.3.1.

The workflow in the IDS can be sketched as follows: the IDS waits for incoming service requests coming from the user. In some cases, these requests may be completed immediately. In other cases, a deferred operation that will be processed in the background later is queued.

Before describing this workflow in detail, we give the definition of the status of a dataset in Section 2. Section 3 lists the service request and what is done in each case. The various operations that are queued for any given dataset are kept track of in a finite state machine. This is detailed in Section 4. When processing the queue, each of the deferred operations is executed in a separate thread. Section 5 describes what is being done for each of the operations. Besides processing service requests, some maintenance task are running regularly in the background. These tasks are described in Section 6. Finally, we provide an index of the context that each storage plugin method is called from in Section 7.

2 Internal states

The processing of a data object is influenced by its internal state and what operations have been queued for them. We say that an operation is *in process* on a dataset if this

Revision: d29d017

operation is either in the queue or if it is currently being executed for that dataset.

The status of a dataset may either be `ONLINE`, `ARCHIVED`, or `RESTORING`. It is `RESTORING` if a `RESTORE` operation on that dataset is in process. If this is not the case, the status is `ARCHIVED` if any other operation then `WRITE` is in process or if the dataset is not empty (e.g. related datafiles exist), but the dataset directory does not exist in the main storage. In all other cases, the status is `ONLINE`.

3 Service requests

In this section, we consider each IDS request and describe in detail what is done in each case. We consider only requests that interact in any way with the storage plugin. The requests `getApiVersion`, `getServiceStatus`, `getSize isReadOnly`, `isTwoLevel`, and `ping` are thus skipped.

Several service calls that deal with a selection of data objects in the storage expect lists of datafile, dataset, and investigation ids as parameter. Most internal processing is done at dataset level and it only matters which datasets are concerned. In these cases, we use the term *selected datasets* regardless whether the objects have actually been selected by datafile, dataset, or investigation id. In a similar manner, we use the term *selected datafiles*.

3.1 `archive`

Queue an `ARCHIVE` deferred operation for each of the selected datasets.

3.2 `delete`

If any of the selected datasets is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for them and throw a `DataNotOnlineException`.

Otherwise delete the selected datafiles from ICAT and from the main storage. Finally queue a `WRITE` deferred operation for each of the selected datasets.

3.3 `getData`

If any of the selected datasets is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for them and throw a `DataNotOnlineException`.

Otherwise get the selected datafiles from the main storage and stream their content to the client.

3.4 `getLink`

If the dataset related to the selected datafile is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for it and throw a `DataNotOnlineException`.

Otherwise get the path of the file from main storage, set an ACL to grant read permission to the user on the file, and create a link to the file.

	ARCHIVE	RESTORE	WRITE
none	ARCHIVE	RESTORE	WRITE
ARCHIVE	ARCHIVE	RESTORE	WRITE_THEN_ARCHIVE
RESTORE	ARCHIVE	RESTORE	WRITE
WRITE	WRITE_THEN_ARCHIVE	WRITE	WRITE
WRITE_THEN_ARCHIVE	WRITE_THEN_ARCHIVE	WRITE	WRITE_THEN_ARCHIVE

Table 1: Update matrix for queued operations. Matrix entries are the result if a new operation (column) is queued while a previous one (row) is still waiting in the queue.

3.5 `getStatus`

Return `ONLINE` if all selected datasets are `ONLINE` (Sec. 2), otherwise return either `ARCHIVED` or `RESTORING`.

3.6 `isPrepared`

If any of the selected datasets (as stored in the previously prepared data selection) is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for them and return `false`.

Otherwise return `true`.

3.7 `prepareData`

If any of the selected datasets is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for them. Store the data selection and other parameter of the request and return the prepared id for later referral.

3.8 `put`

If the dataset referenced in the request is not `ONLINE` (Sec. 2), queue a `RESTORE` deferred operation for it and throw a `DataNotOnlineException`.

Otherwise store the uploaded file in the main storage, create the datafile object in ICAT, and queue a `WRITE` deferred operation for the dataset.

3.9 `restore`

Queue an `RESTORE` deferred operation for each of the selected datasets.

4 Finite state machine

The finite state machine manages the queue of deferred operations for each dataset. The operation may be changed if another operation is queued for the same dataset while the previous one is still waiting in the queue, see Tab. 1 for the update matrix.

The queue is processed regularly by a timer task that starts a new thread to execute each of the pending operations. The execution of a queued operation for a dataset is hold back while a previous operation for the same dataset is currently being executed. `WRITE` operations are processed with a delay that is fixed when the operation is queued. Any subsequent `WRITE` operation queued for the same dataset pushes this delay further.

5 Deferred operations

When the queue of deferred operations is processed, a new thread is started to execute each of them. This section describes what these threads do in each case.

5.1 `ARCHIVE`

Delete the dataset from the main storage.

5.2 `RESTORE`

Get the ZIP file of the dataset from the archive storage, extract the datafiles from it, and store the datafiles in the main storage.

5.3 `WRITE`

If the dataset directory does not exist in the main storage, delete the ZIP file of the dataset from archive storage.

Otherwise create a new ZIP file, get all datafiles that belong to the dataset (according to ICAT) from the main storage and add them to the ZIP file. Store this ZIP file in the archive storage.

5.4 `WRITE_THEN_ARCHIVE`

As the name suggests, same as `WRITE` and then `ARCHIVE`:

If the dataset directory does not exist in the main storage, delete the ZIP file of the dataset from archive storage.

Otherwise create a new ZIP file, get all datafiles that belong to the dataset (according to ICAT) from the main storage and add them to the ZIP file. Store this ZIP file in the archive storage. Delete the dataset from the main storage.

6 Maintenance tasks

There are some maintenance tasks running in the background independently from user actions.

6.1 `FileChecker`

Iterate over all datasets in the ICAT including their datafiles. For each dataset, get the ZIP file from the archive storage and inspect it to check that the list of datafiles in the ZIP file matches the list of datafiles in the dataset and that length and checksum matches for each of the datafiles.

6.2 `Tidier`

Query the main storage plugin for a list of datasets to remove in order to get the overall size of the main storage below the configured limit. Queue an `ARCHIVE` deferred operation for each of the returned datasets.

7 Index of plugin method calls

In this section, we list for each of the storage plugin methods the context that it is called from. Methods not listed here are never called in the considered configuration.

7.1 `archiveStorage.delete(DsInfo)`

Called from the `DsWriter` (Sec. 5.3) and the `DsWriteThenArchiver` (Sec. 5.4) if the dataset directory does not exist in the main storage.

7.2 `archiveStorage.get(DsInfo, Path)`

Called from the `DsRestorer` (Sec. 5.2) to extract the ZIP file from the archive storage into main storage. Called from the `FileChecker` (Sec. 6.1) to inspect the ZIP file.

7.3 `archiveStorage.put(DsInfo, InputStream)`

Called from the `DsWriter` (Sec. 5.3) and the `DsWriteThenArchiver` (Sec. 5.4) to store the ZIP file in the archive storage.

7.4 `mainStorage.exists(DsInfo)`

Called from `IdsBean` while processing various service requests (Sec. 3) to determine the status of a dataset. Called from the `DsWriter` (Sec. 5.3) and the `DsWriteThenArchiver` (Sec. 5.4) to check if the dataset directory exists.

7.5 `mainStorage.exists(String)`

Called from `IdsBean` while processing a `delete` request (Sec. 3.2) to check if a selected datafile exists in the main storage before deleting it.

7.6 `mainStorage.delete(DsInfo)`

Called from the `DsArchiver` (Sec. 5.1) and the `DsWriteThenArchiver` (Sec. 5.4) to delete the dataset from the main storage.

7.7 `mainStorage.delete(String, String, String)`

Called from `IdsBean` while processing a `delete` request (Sec. 3.2) to delete the selected datafiles from the main storage. Called from `IdsBean` while processing a `put` request (Sec. 3.8) to delete the uploaded file from the main storage again in the case that an error occurred while creating the correspoding datafile object in ICAT.

7.8 `mainStorage.get(String, String, String)`

Called from `IdsBean` while processing a `getData` request (Sec. 3.3) to get the selected datafiles from the main storage. Called from the `DsWriter` (Sec. 5.3) and the `DsWriteThenArchiver` (Sec. 5.4) to get the datafiles and add them to the ZIP archive.

7.9 `mainStorage.getPath(String, String, String)`

Called from `IdsBean` while processing a `getLink` request (Sec. 3.4) to get the path of the file from main storage.

7.10 `mainStorage.put(DsInfo, String, InputStream)`

Called from `IdsBean` while processing a `put` request (Sec. 3.8) to store the uploaded file in the main storage.

7.11 `mainStorage.put(InputStream, String)`

Called from the `DsRestorer` (Sec. 5.2) to store the file extracted from the ZIP into main storage.

7.12 `mainStorage.getDatasetsToArchive(long, long)`

Called from the `Tidier` (Sec. 6.2) to query the main storage plugin for a list of datasets to remove.