Chapter 3

Retrieve Workflows

There are three workflows associated with retrieving files:

- PREPARE (§3.1) to stage a file from tape to the EOS disk buffer
- QUERY_PREPARE (§3.2) to query the status of a file (on disk/not on disk, in-flight or not, any errors)
- ABORT_PREPARE (§3.3) to cancel a PREPARE request
- EVICT_PREPARE (§3.4) to remove a retrieved file from EOS disk

The ALICE/JAlien use case may also need to have a more HSM-like behaviour, Implicit Retrieve (discussed in §3.5).

The communication protocol for the above workflows is shown in Fig. 3.1. For most of our use cases, the client is FTS, which sends bulk requests of 200 files at a time. However, the system will also work with `xrdfs` or any client which supports the XRootD protocol. In the case of ALICE/JAlien, each request contains a single file.

Each request consists of a Request ID and a list of files. In the case of PREPARE, the ReqId is provided by XRootD when the request is created. In the case of QUERY_PREPARE and ABORT_PREPARE, this same ReqId should be provided along with the list of files.

EOS does not maintain the mapping between ReqId and the list of files (*i.e.* it is not possible to look up the list of files associated with a given request, but it is possible to query a file to see if it has the

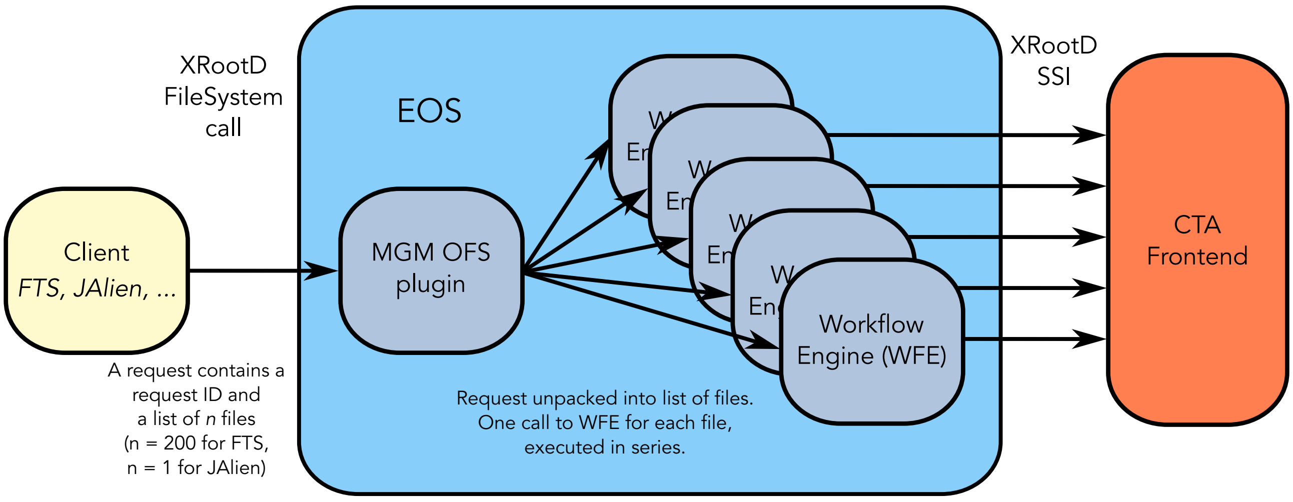
XRootD FileSystem call
Client
FTS, JAlien, ...
A request contains a request ID and a list of *n* files (n = 200 for FTS, n = 1 for JAlien)
EOS
MGM OFS plugin
Workflow Engine (WFE)
Request unpacked into list of files. One call to WFE for each file, executed in series.
XRootD SSI
CTA Frontend

Figure 3.1: EOSCTA Retrieve Workflow Protocols

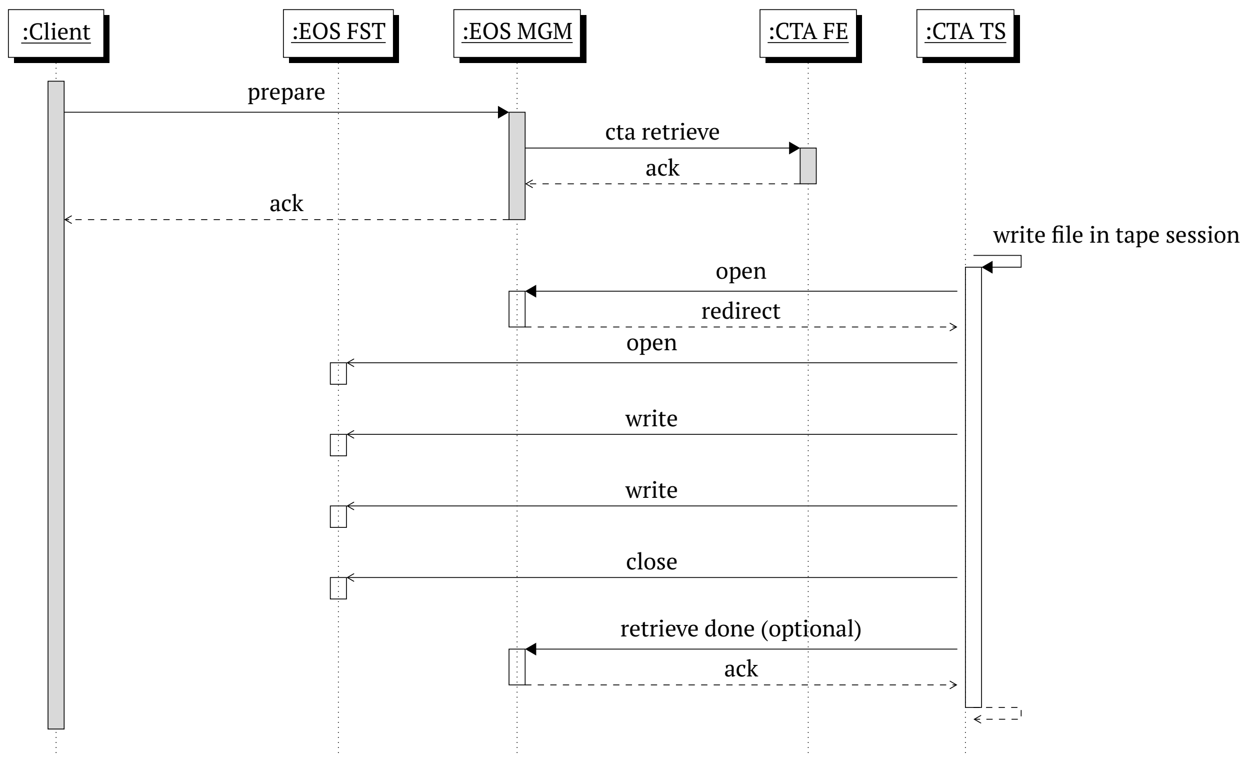

Figure 3.2: File read from tape with the synchronous PREPARE workflow

ReqId attached to it). FTS does maintain the mapping of ReqId to files in its own internal database. For other clients, it is the responsibility of the client to maintain this mapping.

The request is transmitted to the EOS MGM using a standard XRootD FileSystem call. This call is processed in the EOS MGM OFS plugin, where it is unpacked to a list of files (which in the case of JAlien is a list of one element). Each file is dispatched to a handler function in the EOS WorkFlow Engine (WFE). Each file is processed in series. If there is an error for any file in the list, it is skipped over and the next one is processed. Once the list has been processed (successfully or not), the ReqId is returned to the client.

Each call to the handler in the WFE generates a single Google protocol buffer which is transmitted to the CTA Front End across the SSI interface. This call returns a synchronous reply with success or error message for each call.

[If communication overhead is a problem, it would make sense not to split up the batch of requests inside the MGM. Instead we could send through a single request containing the list of files. This is straightforward to implement on the CTA side. On the EOS side it would require changes to how the OFS plugin communicates with the WFE.]

3.1 PREPARE

Fig. 3.2 shows the full workflow for queuing a PREPARE request and retrieving the file to EOS disk.

When the PREPARE request for a file arrives in the WFE, the following steps are executed:

1. Check if the file is on disk, if so take no further action.
2. Check if the `sys.retrieve.req_id` is set:

(a) If yes: there is already an in-flight request. Add the ReqId to the list of requests waiting for this file.

(b) If no: this is the first request for this file. Set `sys.retrieve.req_id` to the ReqId and set `sys.retrieve.error` to empty. Send the PREPARE request to the CTA Frontend. Set `sys.retrieve.req_time` to the time that the PREPARE request was sent.

3. If there is an error: clear the list of requests by setting `sys.retrieve.req_id` to empty and set the error message in `sys.retrieve.error`. `sys.retrieve.req_time` is also cleared.

[In addition to the above extended attributes which are set by the WFE, the CTA Frontend also sets an extended attribute when a PREPARE request is queued. The `CTA_RetrieveRequestId` is set to the address of the retrieve request in the CTA Object Store. This is required for ABORT_PREPARE, because the Object Store has no way to dereference the XRootD request ID into the address of the request object to be deleted.
This xattr should be renamed to start with `sys.` so that it cannot be arbitrarily changed by users. Also the current name is too similar to `sys.retrieve.req_id`.]

[`sys.retrieve.req_time` is set to the last time that a PREPARE request was forwarded to the CTA Frontend. It is intended to be used to detect stale requests (*i.e.* no successful retrieve but no error either). Automatic detection of stale requests is not implemented but this can be checked by an operator.]

3.1.1 retrieve_written workflow

When a file is successfully retrieved from tape, the CLOSEW.retrieve_written workflow is executed. This clears the three attributes `sys.retrieve.req_id`, `sys.retrieve.error` and `sys.retrieve.req_ti`

3.1.2 retrieve_failed workflow

CTA may be unable to retrieve the file, due to an error reading the tape or an error writing the file to the disk buffer. In either case, it will retry three times per session.

If the file cannot be retrieved after two retrieve sessions (six attempts in total), the error is stored in `sys.retrieve.error` as above and the list of pending retrieve requests in `sys.retrieve.req_id` is cleared. `sys.retrieve.req_time` is also cleared.

3.2 QUERY_PREPARE

QUERY_PREPARE allows the client to check the status of files which have been requested by a prior PREPARE request.

[QUERY_PREPARE has not yet been implemented. This section is a proposal for how it should be implemented. See also the GitLab ticket below.]

#648 Implement “xrdfs query prepare” in MGM OFS plugin

3.2.1 STAT vs. QUERY_PREPARE

Currently FTS checks the status of files in a PREPARE request using STAT. The online status of a file is checked using XrdPosixMap::Flags2Mode:

```
if (flags & XrdCl::StatInfo::Offline) *rdv |= XRDSFS_OFFLINE;
```

The `XrdCl::StatInfo::Offline` flag is set if and only if the file has no disk copy (*e.g.*, `d0::t1`).

The error status of a file is checked by checking the `sys.retrieve.error` extended attribute.

There are several problems with using STAT:

- A STAT request queries the status of a single file, so it breaks the normal FTS workflow which is to send requests in batches of 200. QUERY_PREPARE allows FTS to query the status of all files in a request at once.
- STAT is not transparent. Some parts of the PREPARE request status are stored in XRootD flags, other parts are stored in EOS extended attributes. If any of these internal implementation details change, FTS has to be updated. In contrast, QUERY_PREPARE is a standard part of the XRootD protocol. It will provide a single, consistent abstract interface to FTS and any future client software.

3.2.2 Request

The XRootD protocol for QUERY_PREPARE takes a ReqId and an optional list of files as its arguments. As the MGM does not maintain the mapping from ReqId to files, the list of files is mandatory in our case. We can effectively ignore the ReqId as we can query the file using only the filename. However, it is useful to include it in the reply: in the case of asynchronous queries, it allows the client to tie up the reply with the query.

3.2.3 Reply

The XRootD protocol does not specify the format of the reply, other than it is a string. `xrdfs query prepare` simply displays this string on stdout.

As the reply from QUERY_PREPARE needs to be both human-readable and able to be parsed by clients such as FTS, we propose to send the reply in JSON format.

At the time of writing, we are not aware of any other system which implements QUERY_PREPARE. dCache would like to implement it, so we should coordinate with them to provide a uniform reply structure to clients.

The reply should consist of the ReqId, followed by a list of statuses, one per file in the list supplied by the client. Each status should contain:

- The filename
- The file state:
  - file does not exist in the namespace (*i.e.*, bad request)
  - file exists in the namespace and is online
  - file exists in the namespace and is offline
  - optionally: number of tape copies/number of disk copies

- PREPARE status:
  - in-flight (*i.e.*, the file has a valid request ID attached to it)
  - not in-flight (*i.e.*, the file has no request ID attached to it)
  - optionally: confirm that the supplied request ID is attached to the file
  - optionally: list of request IDs attached to the file
- Extra metadata about the request:
  - time that the request was sent to CTA (detect stale requests)
  - error message from CTA if the PREPARE failed

The client logic should be something like:

```
if file state = on disk:
   SUCCESS
elif file state = exists in ns:
   if in_flight:
       WAIT
   else:
       Check error condition, possible retry or report failure
else:
   Unrecoverable error: file does not exist
```

3.3 ABORT_PREPARE

ABORT_PREPARE allows the client to cancel PREPARE requests which are not "interesting" any more. One typical use case (commonly used by ATLAS) is when the PREPARE request for a file is similtaneously sent to CERN T0 and one or more T1 sites. As soon as one of the requests succeeds, the other ones are cancelled.

In CASTOR, the queue for "bring online" is seven days long. Users can cancel "bring online" at any time. CTA follows this same behaviour.

The ABORT_PREPARE workflow is implemented as follows:

1. Check if the `sys.retrieve.req_id` includes the ReqId from the client:
   (a) If yes: the request is still in-flight. Remove the ReqId from the list of requests waiting for this file.
   (b) If no: either the file has been retrieved successfully, an error has occurred, or the client has not provided the correct ReqId. Log what happened and take no further action.
2. If `sys.retrieve.req_id` is now empty, send an ABORT_PREPARE request to the CTA Frontend. Delete the values of `sys.retrieve.error` and `sys.retrieve.req_time`.

3.4 EVICT_PREPARE

The EVICT_PREPARE workflow offers the same functionality as the CASTOR `stager_rm` command. It can be triggered using `eos stagerrm` or `xrdfs prepare -e`.

Whereas ABORT_PREPARE is typically triggered by an experiment's data management software, EVICT_PREPARE is intended for use by operators. ABORT_PREPARE is for cancelling in-flight requests, while EVICT_PREPARE is for removing files which have been successfully retrieved.

[To be added: description of how EVICT_PREPARE works, including which xattrs are updated.]

3.5 Implicit retrieve (open for read)

[The ALICE/JAlien use case may require us to implement an implicit prepare workflow. That is, when a user attempts to open an EOS file which only has a tape copy, EOS should implicitly generate a PREPARE request.
This has not yet been implemented. To be discussed.]