FTS4: Tape daemon re-engineering

João Lopes on behalf of the FTS team

15th September 2026, XRootD & FTS Workshop @ CNRS Rhône Auvergne Delegation Lyon, France

Outline

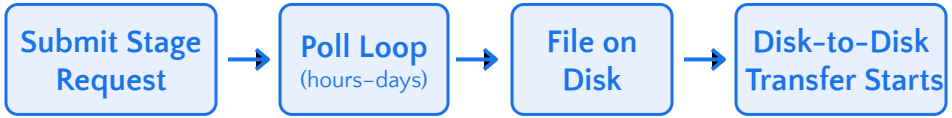
- Importance of tape transfers in FTS
- Challenges in FTS3 w.r.t tape transfers
- The new model for tape transfers in the FTS4
- Current developments in FTS4
- Results from experiment tests
- Future work

FTS, and why tape transfers matter

- **Rising tape adoption in WLCG:** Most transfers are still disk-to-disk, but experiments are increasingly adopting tape as “cheap disk” . ex: ATLAS Data Carrousel model
- **FTS manages data management workflows with tape:**
  - ***archiving*** — validating when files are safely stored to tape media to safely clear disk replica
  - ***staging*** — recalling files from tape back to disk (bring-online), which can take hours or days

Archiving Workflow

Staging Workflow (bring-online)



Tape in FTS3: the “fts-qos” daemon

In FTS3 the “*fts-qos*” daemon handles everything related to tape

Legacy Codebase

- One of the oldest components of FTS codebase.
- Relied heavily on the boost library.
- The component that fewest people understand.

Gfal2 Dependency

- Deeply bound to Gfal2 APIs, dragging unnecessary complexity from the past.
- Shared behavior was forced through excessive use of inheritance which made all the classes entangled.
  - Gfal2Task → BringOnlineTask → PollTask → ...

Rigid architecture

- No separation between scheduling decisions and task execution
  - Lead to suboptimal decisions

/fts-qos

```
Gfal2Task
├── ArchivingPollTask
├── BringOnlineTask
│   └── PollTask
└── HttpBringOnlineTask
    └── HttpPollTask

JobContext
├── ArchivingContext
└── StagingContext
    └── HttpStagingContext
```

Complexity and legacy codebase have blocked feature developments in recent years.

Tape in FTS3: the “fts-qos” daemon

Challenges for operations during RUN3

Inefficient on the DB

- Staging and archiving requests are constructed in batches of files, however all the DB updates are done in one transaction per file inside a for loop

Difficult to trace transfer lifecycle

- FTS3 DB is disk-to-disk transfer oriented
- No DB data structure for tape requests even though that’s the unit of work for the tape daemon

Lack of flexibility on configuration

- Relies on a central configurations preventing storage level tuning: request batch sizes, number of requests, polling frequency etc
- Hard to answer basic questions: How often are we polling? What are the batch sizes? the knobs aren’t visible or tunable

Fragile polling mechanism

- A single failure to poll the status of the request fails files:
  - FTS-2045 Forgiving tape polling
- Daemon restart would recover from the DB all ongoing requests in a single burst

These problems make tape systems inefficient and took valuable time away from the FTS team and experiments data management teams

FTS4: New model for tape transfers

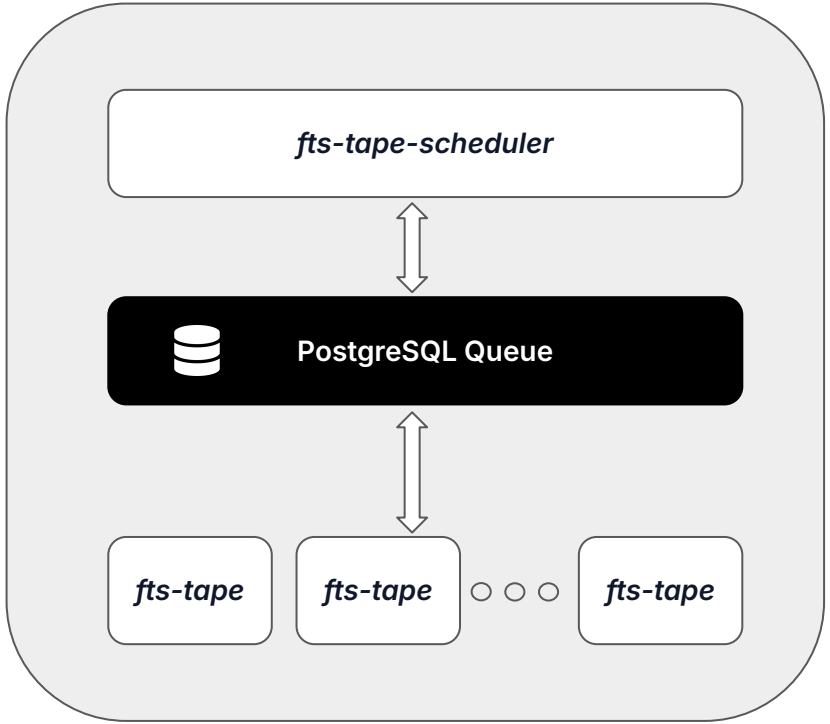
The FTS4 development focused on optimizing DB access and disk-to-disk transfers. But many of the new developments can also be applied to tape transfers:

- New deployment model: Separation between scheduling and worker execution
  - Scheduler is a lightweight process written in Python
  - Worker is a C++ daemon that pulls new request from DB handles HTTP requests to the storages
    - In FTS4 the worker evolved from the old *"fts-qos"* daemon
- Database updates in batch to improve efficiency
- Gfal2 replacement with internal *FileOperations* library
  - Benefiting from data access protocol alignment
    - Sites and Data Management experts agreed on the "WLCG Tape REST API"
- Re-use modernized libraries from transfer daemon

New architecture to handle tape requests

Learning from the disk-to-disk architecture

- **Single Python scheduler:**
  - Elected dynamically via DB
  - Decides scheduling strategies: handles priorities, waiting factors, batch sizing, and per storage concurrency limits
  - All configuration variables are stored in a separate table *t_se_tape_config*
- **Distributed C++ workers:**
  - Runs horizontally on every active host in the FTS cluster
  - Grabs ready requests from database and executes them:
    - Issues HTTP requests, and pushes individual success/failure updates to the DB



Tape scheduler daemon

What it does:

- Leader-election via the heartbeat in the DB
  - High available
  - Makes sure that only one scheduler is active at a time
- Reads configure storage capacity limits, and other config parameters from DB
  - Decides what, when and how much work to has to be done
- Calls a DB stored procedure to create a staging/archiving request containing a batch of files and writes them to the DB

Requests are persisted in the DB

- The new implementation introduces:
  - ***t_staging_request***: stores individual staging requests
  - ***t_archiving_request***: stores individual archiving requests
- On restart FTS3 could reorganize files in different batches
- Allows easier debugging operations

Tape worker daemon (fts-tape)

What it does:

- Polls the database for new requests using the DB as a non-blocking concurrent task queue using:
  - *SELECT FOR UPDATE SKIP LOCKED*
- Each requests becomes a task in an internal priority queue ordered by execution time:
  - Polling the request later is just re-queuing the same task with a delay
- Executes HTTP requests for staging or archiving files
- Collects individual file state changes and pushes them to the DB
  - Asynchronously using bulk updates

FTS4 does not use Gfal2

- All tape related requests go through the new FTS *FileOperations* interface
  - Stage-in, cancel stage-in, poll locality, evict disk file
- Replaced legacy dependencies with HTTP requests

Code harmonization with FTS4

- Database access: Uses common DbApi interface
- Status reporting: “Push to queue and write in batch” pattern like disk-to-disk transfers
- Authentication: Same ZMQ interface to *fts-token*
- Daemon lifecycle: Reuses *libfts_common* to handle heartbeat, signal handling, termination etc

FTS4 Tape Enhancements

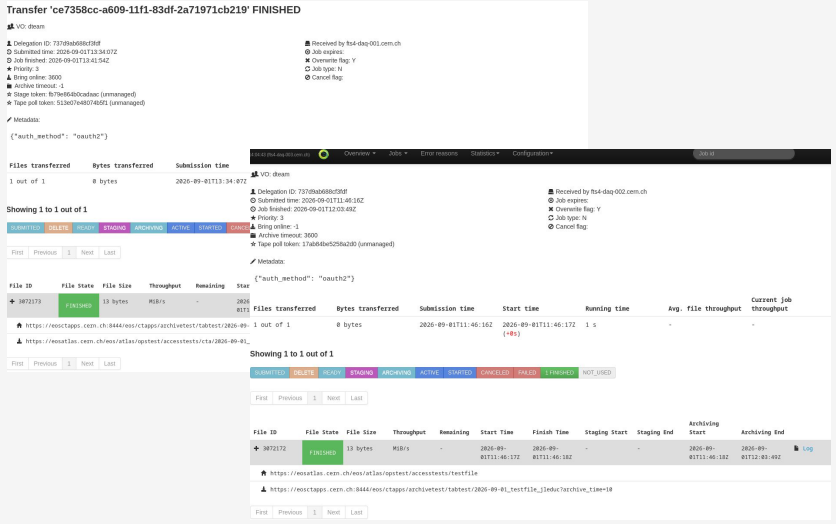
- Tape requests configuration parameters can be set per storage endpoint:
  - Staging and archiving requests sizes, waiting factors, polling frequency, concurrency limits etc
  - Different endpoints can be configured differently out of the box as requested by experiments
  - https://helpdesk.ggus.eu/#ticket/zoom/1002105
- On server restart tasks are spread over the configured polling intervals instead of synchronous polling everything at the same time
  - Jitter was also added to the tasks deadline to mitigate this problem even further

- More resilient polling was implemented
  - Poll only stops when a terminal state is communicated via the specific fields defined in the Tape REST API spec or a configured timeout is meet
- Support for token authenticated tape REST API requests has been added

Tape and token transfers tests

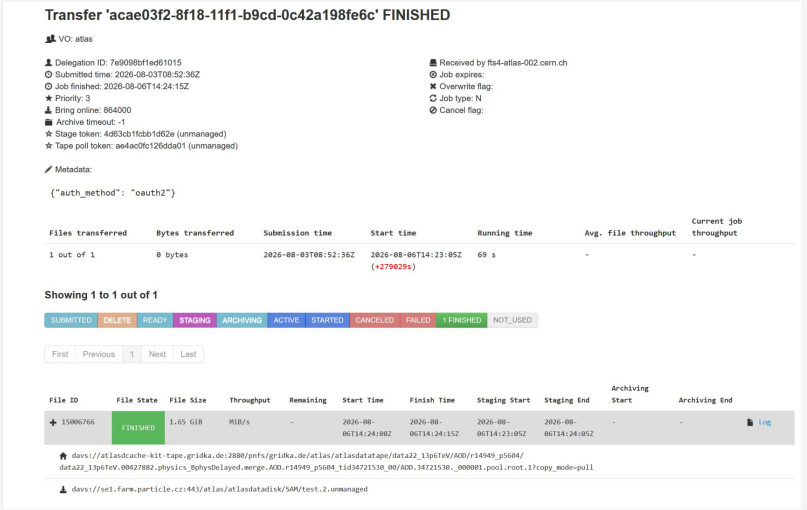
CTA

- The CTA team has successfully archived and staged files to/from CTA using FTS4 and tokens



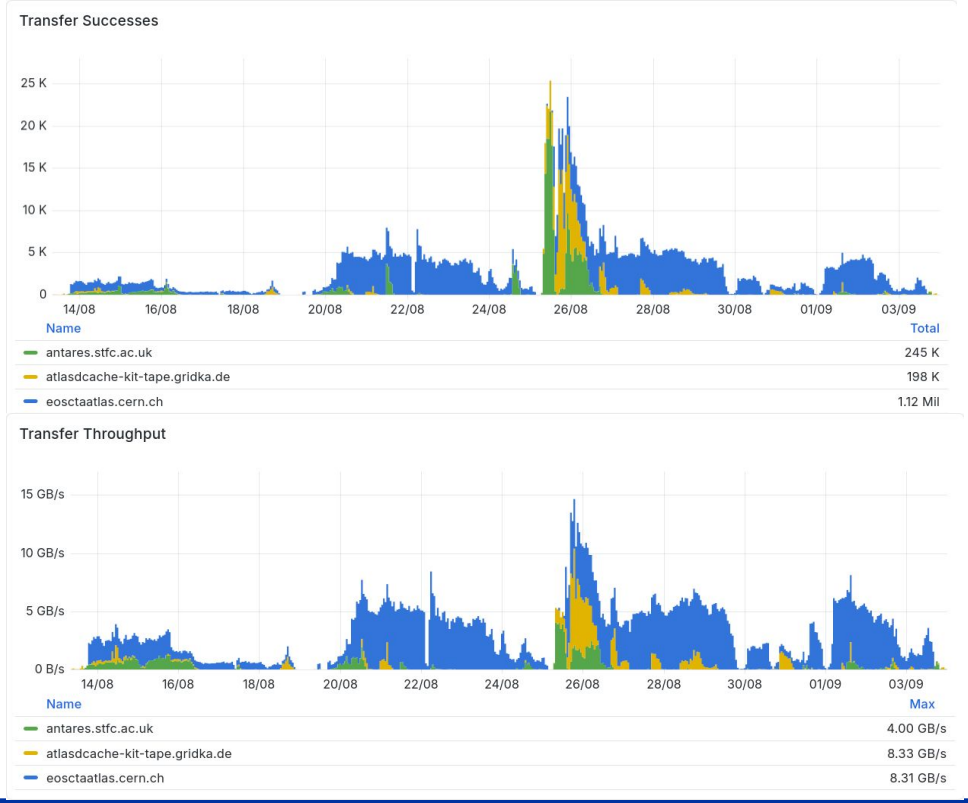
dCache

- ATLAS DM experts have successfully staged files from dCache using FTS4 and tokens



FTS4-ATLAS production tape transfers

- Testing production tape transfers has proven to be challenging
  - Tape workflows are slow and detecting erros is not always easy
- Thanks to ATLAS DM team FTS4-ATLAS managed to get production tape transfers:
  - Between the 14th of August and the 4th of September
  - 3 different tape sites were configured in RUCIO to use FTS4-ATLAS for stage-in transfers



FTS4-ATLAS production tape transfers

Many thanks to the ATLAS Data Management team for their early testing and valuable feedback.
The tests have helped us to identify and fix numerous issues in the first implementation of the new *fts-tape* daemon in FTS4.

- FTS-2503: FTS4: Tape: Put back support for stage-in files from dCache
- FTS-2529: FTS4: Fix regression preventing fts-url-copy from evicting tape files
- FTS-2530: FTS4: Fix regression when matching paths in JSON polling responses
- FTS-2509: FTS4: Tape: Enforce short request timeouts for tape request
- FTS-2509: FTS4: Fix state message timestamps being offset by +2h
- …

Future work

The new tape daemon is deployed on FTS4-ATLAS but some work items need to be finished:

- **Improve UI/UX** - New DB schema allows better tracking and configuration of individual requests but this information still needs to be added to REST API and WEB-UI for user access
- Make logs Tape REST API HTTP calls accessible to DM operators
    - FTS-2092: FTS4: Give end-users access to tape-transfer logs
- Possibly merge the tape scheduler with the disk-to-disk scheduler to reduce code duplication
- Continue to test at scale:
    - We need to know what are the experiment's requirements for DC2027

FTS

File Transfer Service

Thank you!

Questions & Discussion

FTS
File Transfer Service

Backup Slides

New tape storages config parameters

- Allows set limits and config options per storage and vo

t_se_tape_config

Column	Type
🔑 storage	VARCHAR(150)
🔑 vo_name	VARCHAR(100)
archive_batch_size	INTEGER
archive_max_wait_seconds	INTEGER
staging_batch_size	INTEGER
staging_max_wait_seconds	INTEGER
staging_poll_interval_seconds	INTEGER
archive_poll_interval_seconds	INTEGER
staging_concurrent_ops	INTEGER
archiving_concurrent_ops	INTEGER

🔑 **Composite Primary Key** (storage, vo_name)