FERMILAB

FERMI NEWS

A U.S. DEPARTMENT OF ENERGY LABORATORY

Dmitry Litvintsev

20th International dCache user workshop

NIKHEF, Amsterdam, May 6, 2026

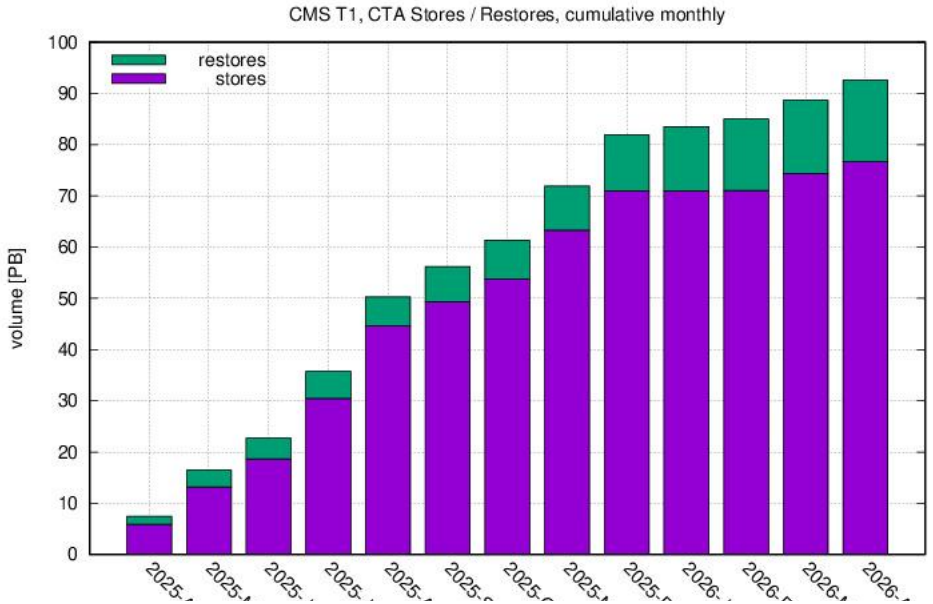
Outline

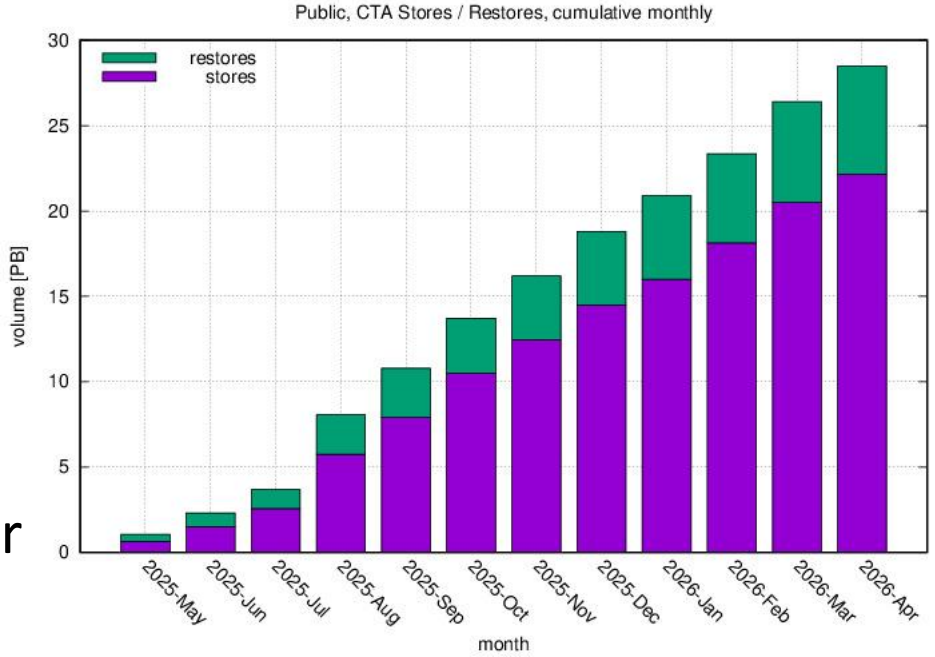
- One year of CTA
- Hot File Replication
  - demo

One year of CTA

```
accounting=# select max(date) from
encp_xfer where storage_group = 'cms';
         max
---------------------
 2025-04-07 07:38:23
(1 row)
```

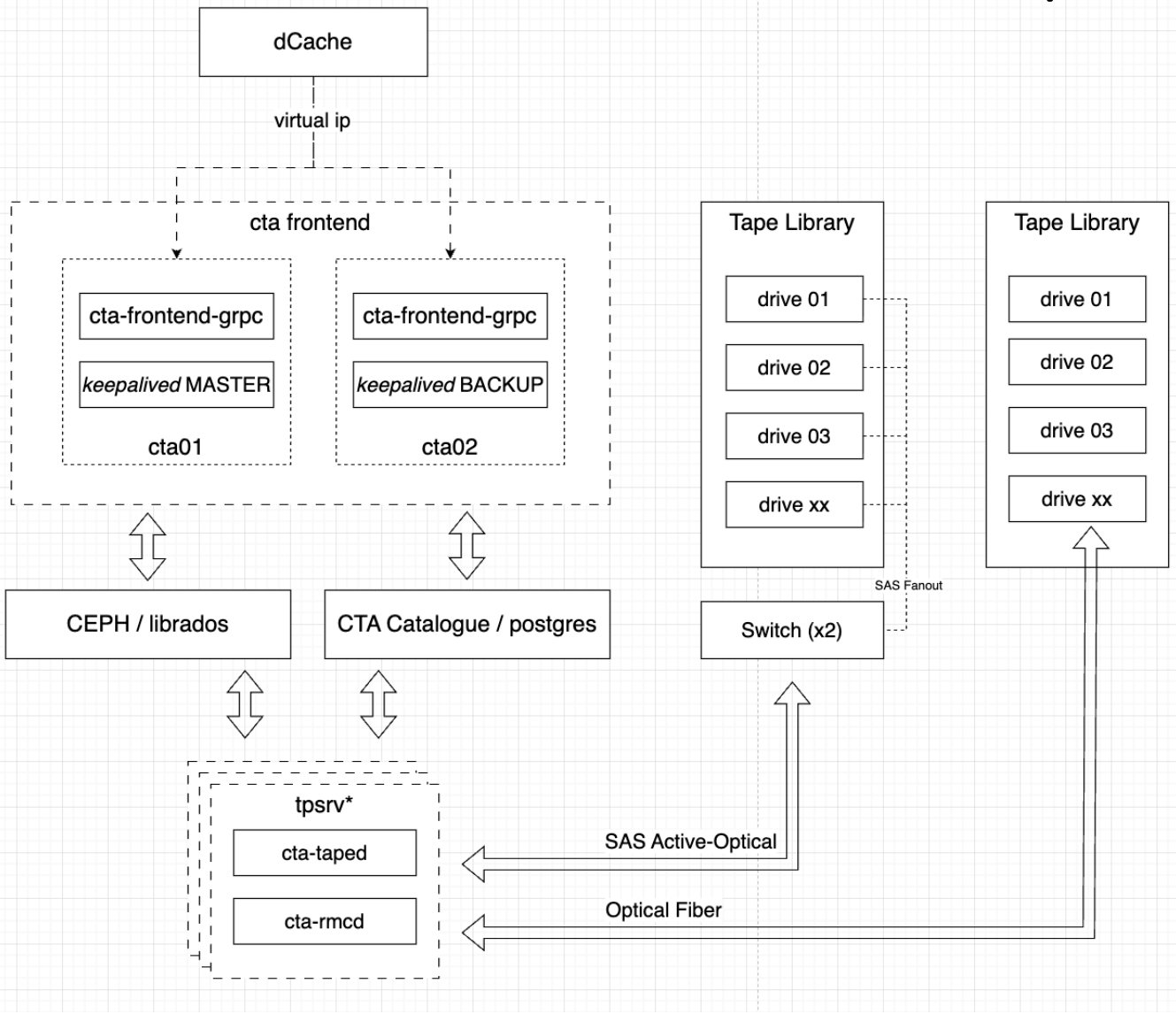
```
accounting=# select max(date) from
encp_xfer where storage_group != 'cms';
         max
---------------------
 2025-05-05 09:56:54
(1 row)
```





Close to 100 PB written into CTA in one year (total currently is 500 PB)

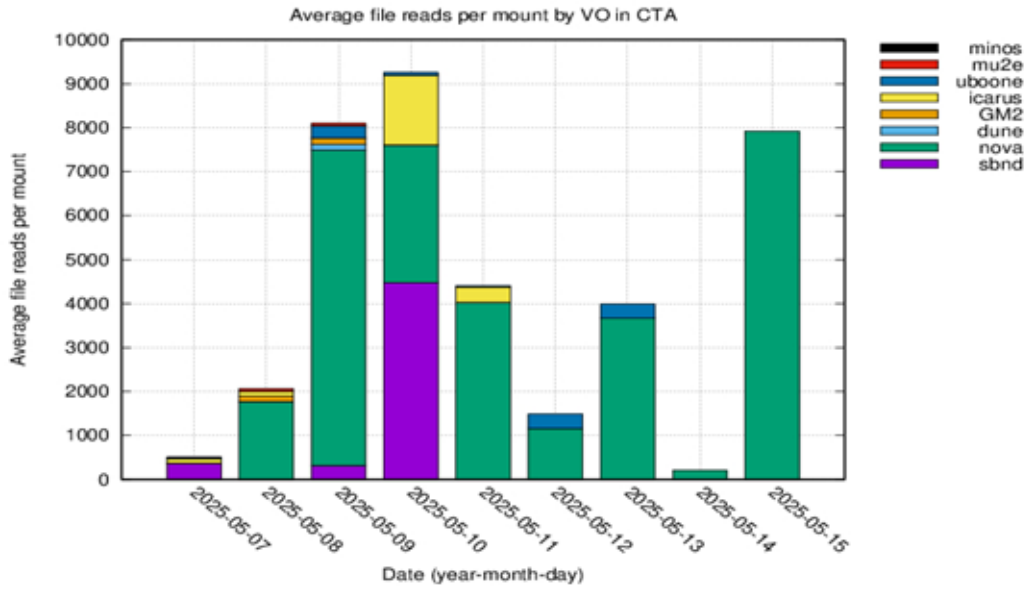
dCache/CTA

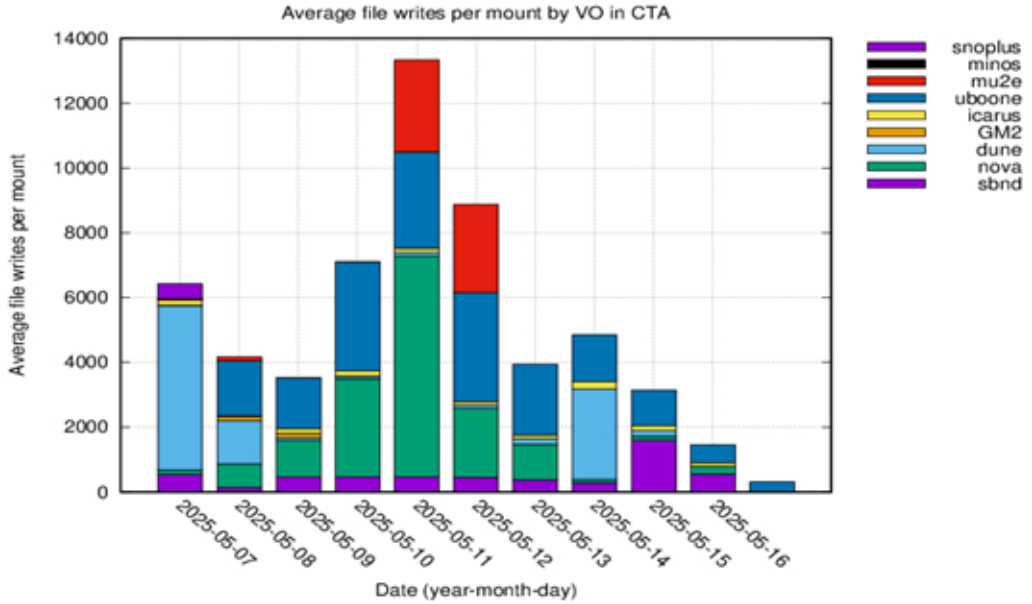


Name	Model	Used slots	Media	
F1	TFinity	11979	6292	LTO8
			5687	LTO9
F2	TFinity	8788	8788	LTO9
F3	TS4500	7698	3601	LTO7M
			4097	LTO8
G1	TS4500	9036	3720	LTO7M
			5316	LTO8
G2	TS4500	9728	171	LTO7M
			9557	LTO8
G3	TFinity Plus	300	300	LTO10

CTA benefits

- Significantly improved tape access efficiency.
  - dCache pools no longer queue stores/restores.
  - Passthrough to CTA:
    - All scheduling is on CTA end only.
    - Unlimited CTA queue allows to sort huge number of requests by volume, location to minimize tape mounts and seeks. Can be further tuned by mount policies per VO.



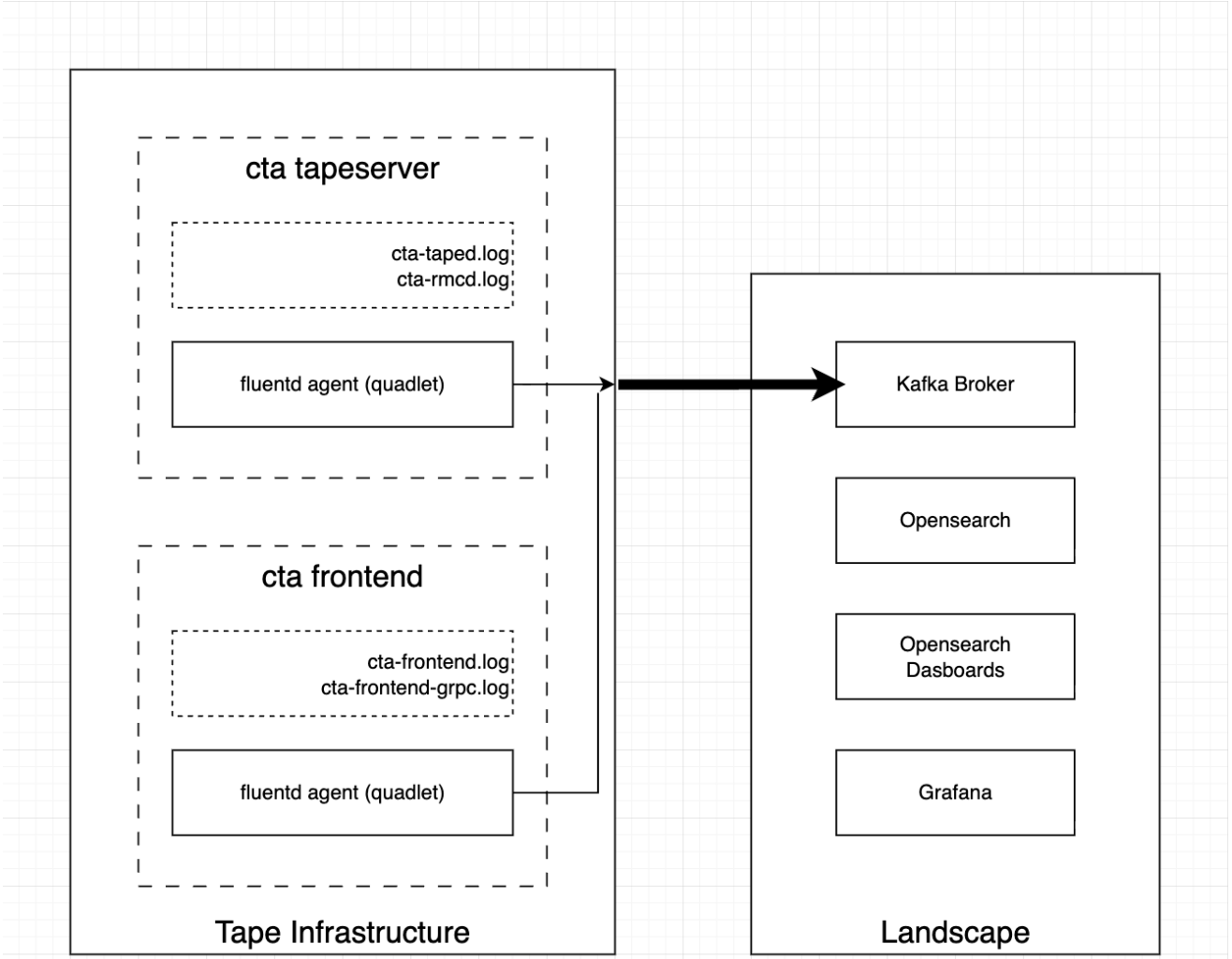


CTA benefits

- Decreased mounts and seeks => less frequent EOL tapes.
- Improved transfer rates:
  - CTA reads / writes files in groups => ”transfer setup” time is amortized.
    - Writes occur at (almost) drive rate.
  - File block location and RAO greatly improve read performance.
    - Enstore written tapes do not benefit from that and Enstore tapes are read with ”Enstore speed”.
- dCache pool node health:
  - When using Enstore, each store/restore request spawns a process (HSM script) holding a connection, resulting in hitting file descriptor limits and high memory consumption.
  - With CTA dCache uses dcache-cta driver which implements NearlineStorage interface and communicates w/ CTA via gRPC. *Only one connection open*. And smaller memory footprint.

CTA monitoring

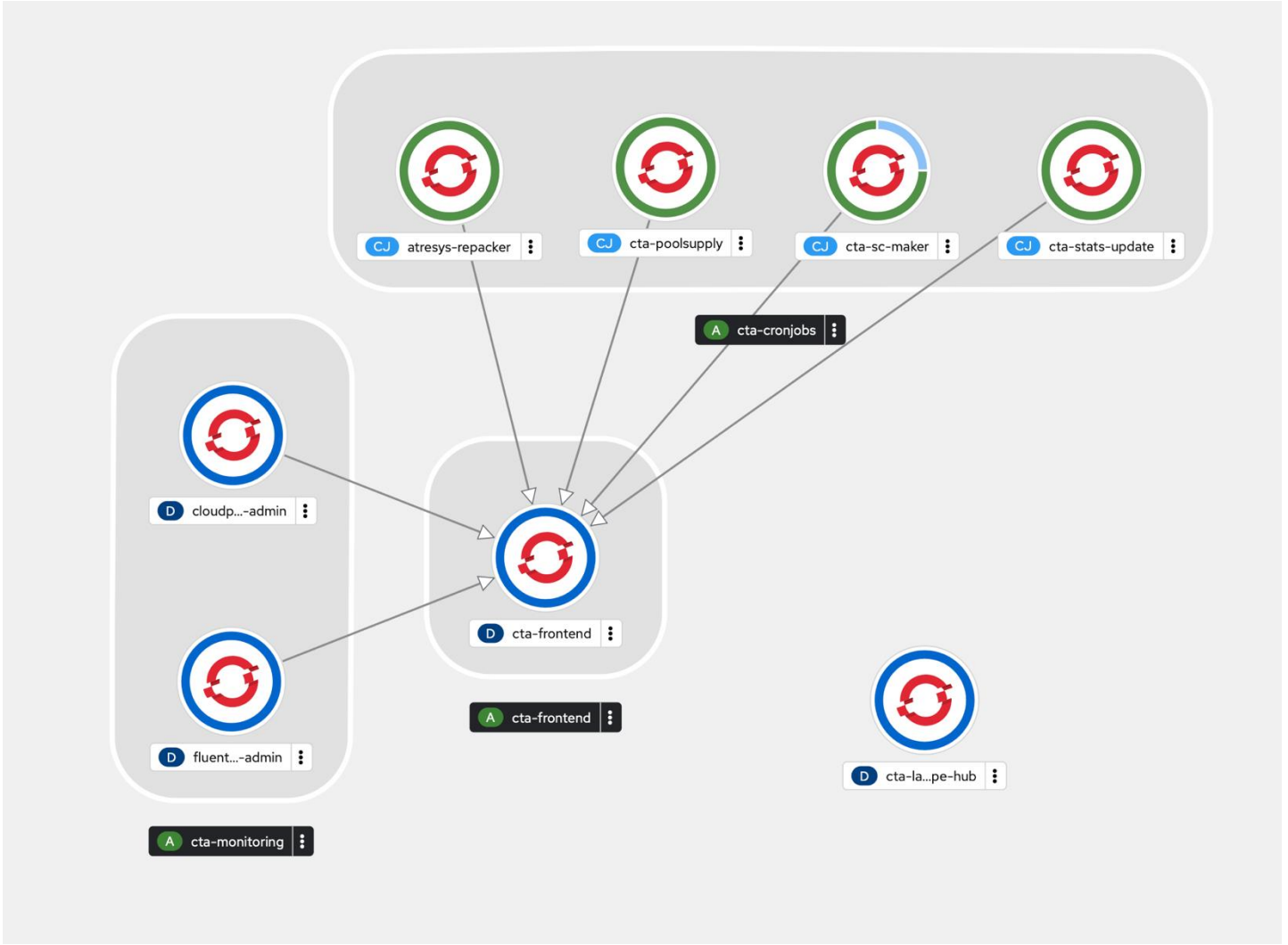
- Unlike Enstore, CTA has no built-in monitoring.
- Fermilab tape operations group developed monitoring pipeline based on *fluentd* watching CTA log files and sending messages to a Kafka server. CTA outputs log messages in json format which helps.
- Opensearch/Grafana dashboards.



https://landscape.fnal.gov/

CTA monitoring and aux tasks on OKD

- OKD cluster at Fermilab is used to host CTA frontend and monitoring components deployments on Kubernetes.
- Runs important crons:
  - Atresys repacker.
  - poolsupply – mimicks file family width behavior.
  - sc-maker – register newly created storage classes in CTA.
  - sta-stats-update – updates tapes aggregate numbers.
- Configurations of these services is in private Fermilab GitLab Repository by can be provided on demand (my understanding).



dCache/CTA problem points

- dcache-cta driver runs a standalone xrootd server, one server per pool.
- When storing/restoring file dCache sends archive/retrieve request, and a file handle (xrootd URL) to CTA.
- CTA replies with unique request ID and puts it into its queue.
- dCache does not persist CTA request ID.
- Then, nothing happens until cta taped pulls request from the CTA queue. This could take hours depending on the CTA mount policy.
- If a pool goes down taking all store and restore requests with it, CTA knows nothing about it ending up with “orphan” requests which it will keep retrying (seemingly forever) and failing due to file handles no longer available.

dCache/CTA problem points (cont)

- These retries are not harmless:
  - Each time the tape will be mounted and positioned and a transfer attempt will be made only to discover that file handle can’t be connected to.
- CTA works with groups of files. When archiving, there could be significant time between file transfer and its registration in CTA catalog. dCache does not receive reply that file is archived until the whole group is registered. If pool goes down before CTA replied, the pool will **flush** file again after restart.
  - This situation creates a “poison pill” on CTA end. It will keep trying to put that file to tape as a part of a group of other files and fail the whole group wasting a lot of resources and not advancing.
- Other failure modes having to do with CTA components restarting or being “stuck” that lead to similar outcomes.

dCache/CTA problem ~~solution~~ mitigation

- I have developed a package pydcache where I plan to collect various dCache or dCache/CTA helper or maintenance scripts developed over the years and scattered across many repos.
- https://github.com/DmitryLitvintsev/pydcache
- Specifically https://github.com/DmitryLitvintsev/pydcache/blob/main/scripts/cta_nanny.py

  is used to subscribe to Kafka broker to receive CTA log messages looking for a specific “duplicate key value violates unique constraint” error messages and taking corrective actions:
  - furnishing dCache files w/ HSM locations in chimera
  - declaring file replicas on pools CACHED and
  - cancelling corresponding ACTIVE **stores**.

dCache/CTA problem ~~solution~~ mitigation

- Periodically scan taped logs for “Request not found" and expunge them from CTA queue to prevent them from being retried.
  - Will be rewritten to use Kafka consumer.

dCache/CTA problem - undelete

- When we switched from Enstore to CTA we lost ability to recover metadata of deleted archived files.
- The problem with file recovery is that it can be only recovered by file path, but CTA does not store file path and by the time a pool calls `NearlineStorage.remove` only pnfsid is known (and HSM location).
- I have developed an undelete procedure that relies on a trigger that stores full file paths in Chimera DB on delete.
- Then a script that uses deleted file name restores database entries in both chimera and CTA databases

https://github.com/DmitryLitvintsev/scripts/blob/master/python/dcache/cta/recover_file.py

(the script will eventually be moved to pydcache package)

- During developers workshop following this users workshop we will discuss possible implementation of file undelete that does not involve external scripts.

Summary for dCache/CTA

- Works well, significant functional and operational improvement over Enstore.
- Edge cases need some babysitting:
  - Infrequent “duplicate key value violates unique constraint” errors impact stores.
  - Multiple retries on ”Request not found” errors.
- May be when pools are restarted or go down they should try killing all stores and restores automatically.
- May be CTA requests should be part of file (or pool repo) metadata.
- Need for undelete of tape resident files.
- Continuous monitoring development.

Hot File Replication

- Current Hot Pool Replication works as follows – if pool cost gets high all new requests for file replicas in this pool will trigger p2p transfers.
  - Typically this just does a double whammy on that pool that only gets hotter.
- Chris Green at Fermilab has developed “Hot File Replication” that triggers replication of a file if number of requests to this file exceeds a threshold.
  - Extends existing Migration Module.
  - This feature is now available in 11.2 release.
  - By default the functionality id disabled. Will be ON in the next golden release.
- https://www.dcache.org/manuals/Book-11.2/rf-cc-pool.shtml

Here be Hot File Replication Demo

How to enable/control HFR on pools

```
#
#   Whether to enable hot file replication.
#
(one-of?true|false)pool.hotfile.replication.enable = false

#   The number of cached replicas to maintain for hot files.
#
pool.migration.hotfile.replicas = 1

#   The number of requests required to trigger hot file replication.
#
pool.migration.hotfile.threshold = 50

#   The default concurrency for migration jobs.
#
pool.migration.concurrency.default = 1
```

HFR setting on pool (will take precedence)

```
hotfile set replicas <value>  # Set hot-file
replication parameter.
hotfile set threshold <value>  # Set hot-file
replication parameter.
```

HFR commands on pool

```
> help hotfile
hotfile get replicas # Get hot-file replication parameter.
hotfile get threshold # Get hot-file replication parameter.
hotfile set replicas <value>  # Set hot-file replication
parameter.
hotfile set threshold <value>  # Set hot-file replication
parameter.
hotfile show # Show hot-file replication status.
```

HFR features

- If the number of required replicas is less than the number of available pools – there will be as many replicas made as there are available pools.
- HFR utilizes PoolManager selection mechanism. If there are many pool groups where the file can be read from then the pools from the pool group with the highest readPref will be chosen as replication targets.
  - When making selection decision the the IP and protocol of the transfer are taken when performing network and protocol unit match.
- Currently HFR does not take into account target pool CPU cost, the pool selection is weighted (by available space) random.

Future directions of HFR

- Add CPU cost to pool selection
- We hope HFR can be used as one of the building blocks of self-healing (in so far as hot spot mitigation goes) dCache that can be part of MAPE AI assisted feedback loop.

The End