CERN
Tape Archive

CERN Tape Scheduling Systems

(walk through history and discussion)

Jaroslav Guenther (IT-SD-TAB)

31/S-027

14.05.2024

SHIFT Tape Scheduling History

(~1990) SHIFT

8 years later: STK robot libraries

LEP experiments owned

- tape drives for DAQ tapes
  - trucks moved them to IT Valut
- **namespace catalogue**
  - it had the tape info per file !
  - based inherently on IBM 3480 VIDs !
  - VID vs Run.Event file map

CERN IT

- migration to new denser media → new tape identifier (VID.FSEQ)
- added Tape Management System (TMS):
  - first use of "namespace" to map old to new VIDs (experiment catalogue bridge)
  - enabled repack & writing to new tapes

scheduling mostly done by end users !
(manually 24/7)

CASTOR Tape Scheduling History

next ~23 years: CASTOR (ref.: 2007, 2015)

New LHC experiments:

- agreed that CERN IT maintains the namespace
- **Implications of maintenance-free decision underestimated**

CERN IT introduced

- CASTOR namespace & disk staging area
- scheduling = = **3 daemons deployed centrally**
  - **Stager = FIFO queueing from NS**
    - disk cache management **(& UI)**
    - req. mount while ignoring tape state & location (→ **ops issues**)
  - **VDQM = requests to tape drives**
    - ignored file-level info
    - **danger of disjoint placement of related files across different tapes**
  - **VMGR (previously TMS)** = tape choice

CASTOR Operational Experience

Experiments

- **lost control** over: file location on tape and scheduling
- **additional files transfers** needed:
  - staging area → AFS/EOS **(instead of tape mainframe → analysis machines)**
- no tape-targeted file collocation → **less efficient readout**:
  - repack exercise makes file spread worse (still the case today)

Mitigation Strategies

- **experiments helped the scheduler by**
  - sending large file lists as requests and ordering these by tape (e.g. ATLAS Tier-0 Ops Run I & II)
- **Implemented concepts of**
  - TapePool per VO
  - StagerClass (to keep files separate)
  - inherent FIFO creation time collocation
- **Operational oversight**
  - VO fair share
  - priority management (Stager prone to use busy lib instead of idle one; artificially imposing # tape drives to be used by supply pool logic; ... etc.)

CASTOR Limitations and CTA Implementation

(2022 - present) CTA (ref.: 2017, 2021)

CASTOR blockers:

- memory resident namespace **did not scale**
- **disk resource management** offloaded to LSF scheduler
- centralised scheduling **struggled with load**
- Stager queueing into VDQM "blind" of phys. lib/drive state **suboptimal resource management**
- ... **unmaintainable code** base etc.

CTA implementation:

- **EOS** namespace & disk buffer management
- disk throughput shaping via VO **dedicated buffers per use-case**
- refactored the code into separate **multi-threaded distributed daemons** !
- removed centralised Stager: → **scheduling at mount time**

CTA Scheduler (WIP / TBD)

File Collocation

- **Experiments demand efficient readout**
  - not all care about namespace anymore !
- **CTA "smart" writing to tape:**
  - natural pushback from experiments to control scheduling aka "know where their file is"
  - **archival metadata hints**
    (suggesting what does belong together)
    note: we have StorageClass
    (= what shall be kept separate)

Repack vs Production

- separating scheduler backend
  and separate drive allocation in the pipeline

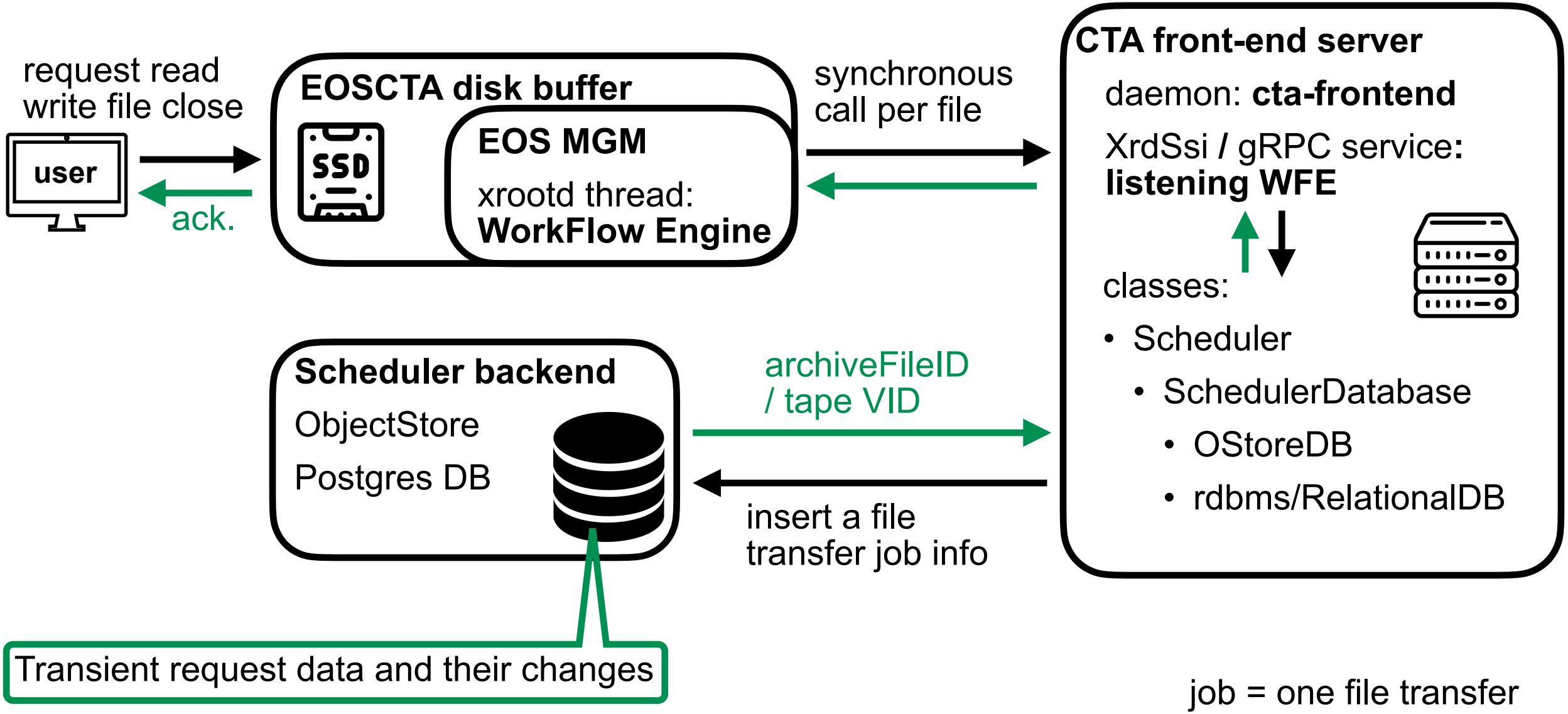
What about ...
good ideas from CASTOR 2007 ?

Automatic assignment of drives per VO

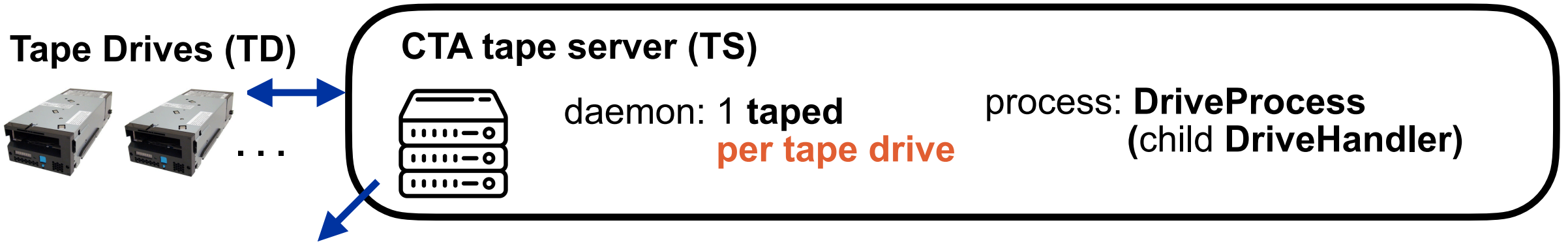
- "proper monitoring of the disk status which must be fed to the scheduling system"

[... add on ...]

CTA Receiving a Request



CTA Tape Server polling 1/2



DataTransferSession forked for a free drive (UP)

- tries to get new (/its own) Mount
- **Mount** = drive assignment to tape for set of jobs
- calls **Scheduler → getNextMount[-DryRun]()**
  - **SchedulerDatabase → fetchMountInfo()**

improve perf **if needed**

- look up only TD relevant info
- lock only what needs to be locked

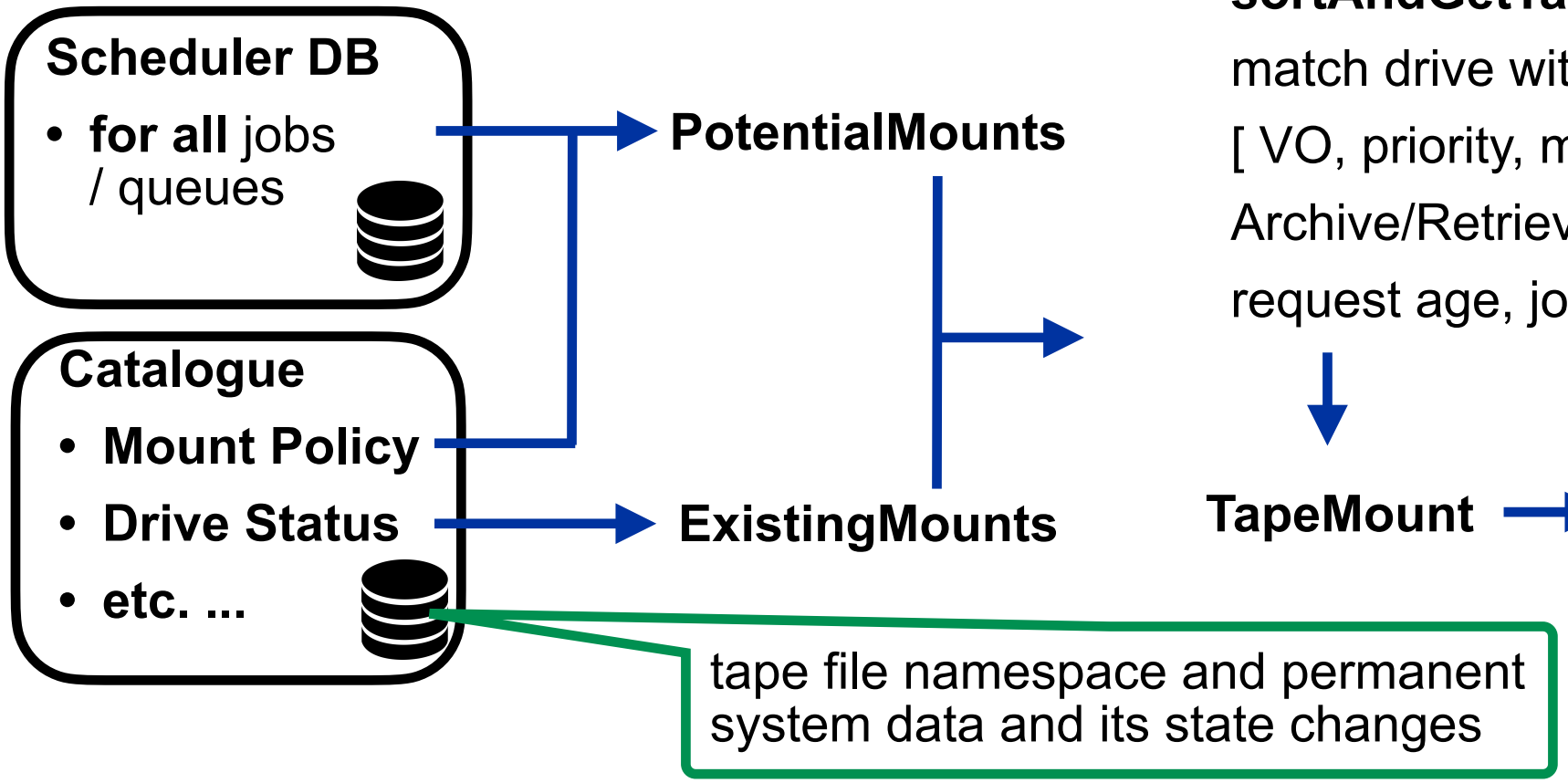
Each *taped* looks at **all jobs** for **all drives**
to get **all** (existing/hypothetical) **Mounts**
+ iterates through → match drive with tape and job set
(1st w/o global lock [-DryRun] + 2nd time with)

job = one file transfer

CTA Tape Server polling 2/2

DataTransferSession

- getting Mounts by polling Scheduler DB and Catalogue
  - **Scheduler → getNextMount[-DryRun]()**



Scheduler.
sortAndGetTapesForMountInfo()

match drive with

[VO, priority, mount type

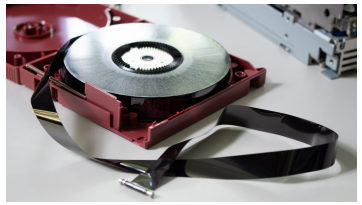
Archive/Retrieve/..., tape VID,

request age, job summary statistics]

TapeMount → **SchedulerDatabase**

updates
mountID, VID per job

CTA Tape Drive with Mount



DataTransferSession

- calls **executeWrite/Read(TapeMount)**
  - several threads are spawned taking care of:
    - **mounting** the **tape**
    - **polling Scheduler DB** for job/queue batches
    - **inserting** the **jobs** to for the execution
    - the **R/W** from/to memory/tape/disk buffer
    - **MigrationReportPacker** thread reporting back to CTA disk buffer (EOS) **(TBD for PGSCHED)**

threads handle
work of any tape drive

aka object ownership
concept in ObjectStore

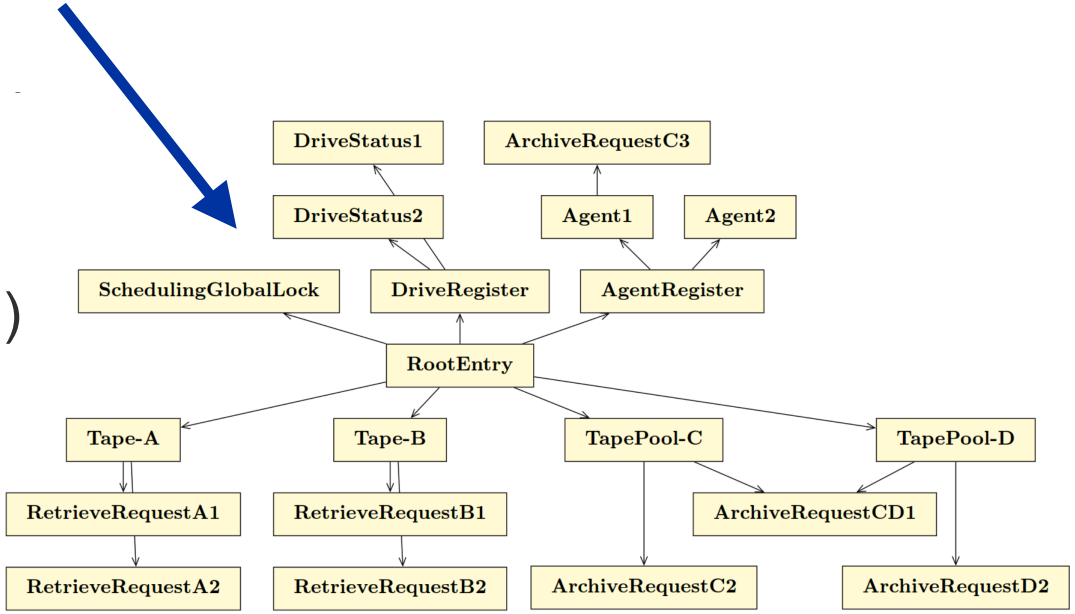
Consistency & Error Handling (TBD for PGSCHED)

- TapeDaemon/**MaintenanceHandler** x-check job "heartbeat" in Scheduler DB
- Scheduler DB "view" on active [VID + mountID] → DriveState check (in the Catalogue) ?

CTA Scheduler ObjectStore

Implementation

- Protobuf serialised objects in key/value ObjectStore
- designed for **performant FIFO queue**
- full **locking** support dev **required**
- **manages** "backpointers" and dangling **pointers**
- **scales well** (despite > storage round trips than DB)
- **multi-threaded interface to ObjectStore**



Intentional complex code development

- inherently ensures **high performance and scaling** (re-inventing a wheel of the DB logic)
- requires **extensive** continuous **learning effort**
- **challenge** for small, high-turnover team at CERN **!**

CTA Scheduler Relational DB

Implementation

- **workflow oriented** tables, views, sequences
  - file transfer **jobs** (Archive/Retrieve/Report/...)
- inherently **uses DB features**
  - facilitates **any job ordering** (FIFO/non-FIFO) locking & MVCC, indexing (+sync), B trees, etc.
- **connection pools** from our *rdbms* wrapper layer
- currently **single threaded interface to DB**

cta_scheduler	
schema_version_major	numeric(20) NOT NULL
schema_version_minor	numeric(20) NOT NULL
next_schema_version_major	numeric(20)
next_schema_version_minor	numeric(20)
status	varchar(100)
is_production	bpchar(1) NOT NULL

archive_job_summary	
mount_id	int8
status	public.archive_job_status
tape_pool	varchar(100)
mount_policy	varchar(100)
jobs_count	int8
jobs_total_size	numeric
oldest_job_start_time	int8
archive_priority	int2
archive_min_request_age	int4

archive_job_queue	
job_id	bigserial NOT NULL
archive_reqid	bigserial NOT NULL
status	public.archive_job_status NOT NULL
creation_time	int8
mount_policy	varchar(100) NOT NULL
vid	varchar(20)
mount_id	int8
start_time	int8
priority	int2 NOT NULL
storage_class	varchar(100)
min_archive_request_age	int4 NOT NULL
copy_nb	numeric(3)
size_in_bytes	int8
archive_file_id	int8
checksumblob	bytea
requester_name	varchar(100)
requester_group	varchar(100)
src_url	varchar(2000)
disk_instance	varchar(100)
disk_file_path	varchar(2000)
disk_file_id	varchar(100)
disk_file_gid	int4
disk_file_owner_uid	int4
archive_error_report_url	varchar(2000)
archive_report_url	varchar(2000)
failure_report_log	text
failure_log	text
is_reportdecided	bpchar(1)
total_retries	int2
max_total_retries	int2
retries_within_mount	int2
max_retries_within_mount	int2
last_mount_with_failure	int8
total_report_retries	int2
max_report_retries	int2
tape_pool	varchar(100) NOT NULL

archive_job_reports	
job_id	bigserial NOT NULL
status	public.archive_job_status NOT NULL
creation_time	int8
mount_id	int8
start_time	int8
priority	int2 NOT NULL
storage_class	varchar(100)
copy_nb	numeric(3)
size_in_bytes	int8
archive_file_id	int8
checksumblob	bytea
requester_name	varchar(100)
requester_group	varchar(100)
disk_instance	varchar(100)
disk_file_path	varchar(2000)
archive_error_report_url	varchar(2000)
archive_report_url	varchar(2000)
failure_report_log	text
failure_log	text
is_reportdecided	bpchar(1)
total_retries	int2
max_total_retries	int2
retries_within_mount	int2
max_retries_within_mount	int2
last_mount_with_failure	int8
total_report_retries	int2
max_report_retries	int2
tape_pool	varchar(100) NOT NULL
repack_filebuf_url	varchar(2000)
repack_fseq	numeric(20)

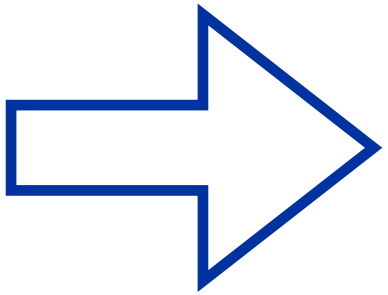
Intentional straightforward code development

- ensures **high performance IF** DB features exploited smartly (e.g. do not ask to count rows, write pop/delete counters)
- **requires optimisation** efforts per use-case
  - relies on the dev diligence with DB queries and DB admin tuning

CTA Scheduling Operations last year

ObjectStore Experience

- **fire-fighting**
  - 5 high priority dev tickets created in the last year
    - object **deletion** #309
    - empty shard **handling** #500
    - infinite **loops** #602
    - **locking** issues #460
    - **repack** exhausting OStore **resources** #573



Relational DB

- **challenges**
  - non-FIFO **priority queues**
  - object structure **("schema") updates**
  - CTA Scheduler code logic **not easy to extend/modify**
    tailored to ObjectStore backend structure (handling object dependencies)

CTA Request Ingestion

Each *taped* looks at all jobs for all drives
to get all (existing/hypothetical) Mounts
(+ 1st w/o global lock (DryRun) 2nd time with)

ObjectStore

- **summary objects** including regularly updated counters
- **locking** + multi-threaded access
- EOS MGM → cta-frontend **ingestion** one by one

Relational DB

- table views and counters
  (counters to be implemented if needed, we can avoid counting rows in queries)
- MVCC, explicit table/row locks, advisory locks
  (more about this later ...)

  smart locking might save us the DryRun
- **idea of bulk inserts if needed**
  (hold set of WFE requests until all in DB)

Postgres DB Management Challenges

MVCC (Multi-Version Concurrency Control)

- consistent "snapshot" views
- keeps all row versions until the oldest active transaction or next automatic vacuuming

Power cut & Recovery

- Incomplete transactions and vacuuming may cause long lockdowns (~1 hour) to replay WAL
- risk of data inconsistency or corruption, prevention

DBOD

- ideal for performance testing, realistic latency (RTT)
- ensure SSDs are used to avoid random access issues

& beware of implicit transactions without auto-commit keeping all history !

PostgreSQL config options:

```
---------
Write-Ahead Log (WAL) settings

wal_level = replica
synchronous_commit = on
wal_sync_method = fsync
---------
Checkpoints

checkpoint_timeout = 5min
checkpoint_completion_target = 0.7
max_wal_size = 1GB
---------
Point-in-Time Recovery

archive_mode = on
archive_command = 'cp %p /path/to/archive/%f'
---------
Streaming Replication

wal_level = replica
max_wal_senders = 5
wal_keep_segments = 32
---------
Autovacuum  (for MVCC cleanup)

autovacuum = on
autovacuum_naptime = 1min
autovacuum_vacuum_threshold = 50
autovacuum_analyze_threshold = 50

... etc.
```

Management of Completed Job Records

Vacuuming

- table scan + version replay + row deletion + reindexing
- **gradually reclaims disk space**
- **slower**, can lock large tables (especially: VACUUM FULL)

Double Buffering + Truncate Table

- use two identical tables, switch between them
- avoids extended lock periods during maintenance
- consistent data access
- Truncate obsolete table:
  - no table scan or history replay checks
  - **fast** row removal
  - **immediately reclaims disk space**

Tape Drive Efficiency and Data Integrity

Tape Free Space

- vendor tape **raw capacity understated**
  - by 1-5%, ~450 GB (?), *stable over time or decreasing ?*
- tape drive writes until **hitting tape end** !
  - **flush** tape writes in **bunches** of 200 files / ~32 GB (hard-coded)
  - last incomplete batch → failure; *time spent writing today **?***
- **cost-effective** (tape is cheap and drives fast today)
  - "waste" max space and time writing 32 GB per tape << extra free space

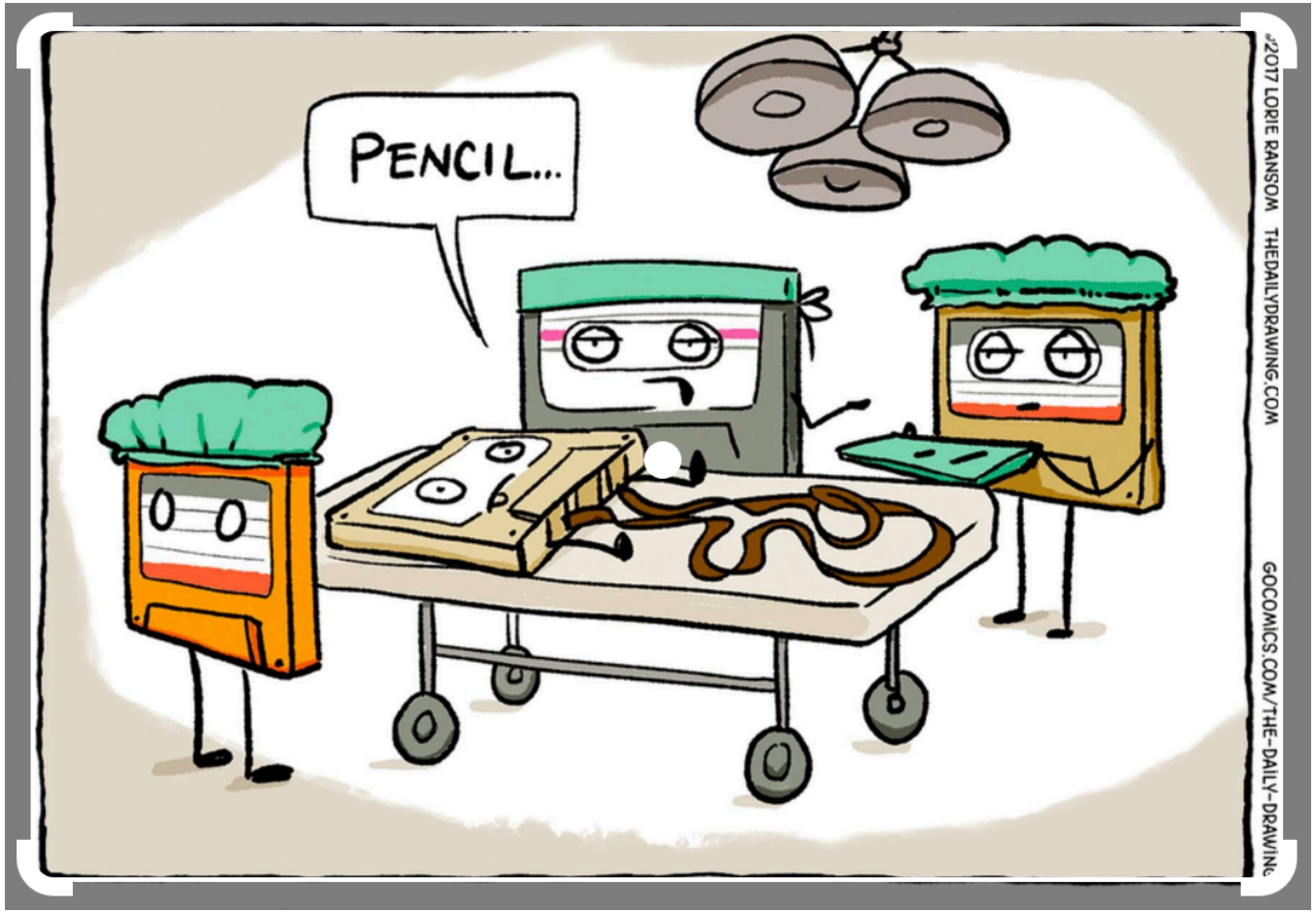
Tape Head Position Check

- there is a SCSI command to query tape drive position
  - avoids unnecessary flushes (Eric's idea)
  - IBM's approval needed to confirm read head position is indicative of what the write head wrote !

Fine-tuning not worth the effort today !

summer student study ?

Thank you for your attention



... and your help !

home.cern