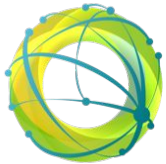
FTS & XRootD Workshop

CNRS Rhône Auvergne Delegation, Lyon – 14-18th September 2026



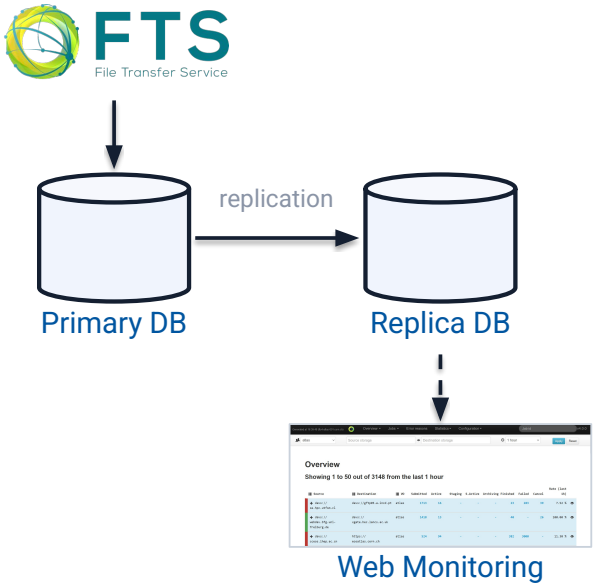
FTS4: Introducing TimescaleDB

Mihai Patrascoiu
on behalf of the FTS team

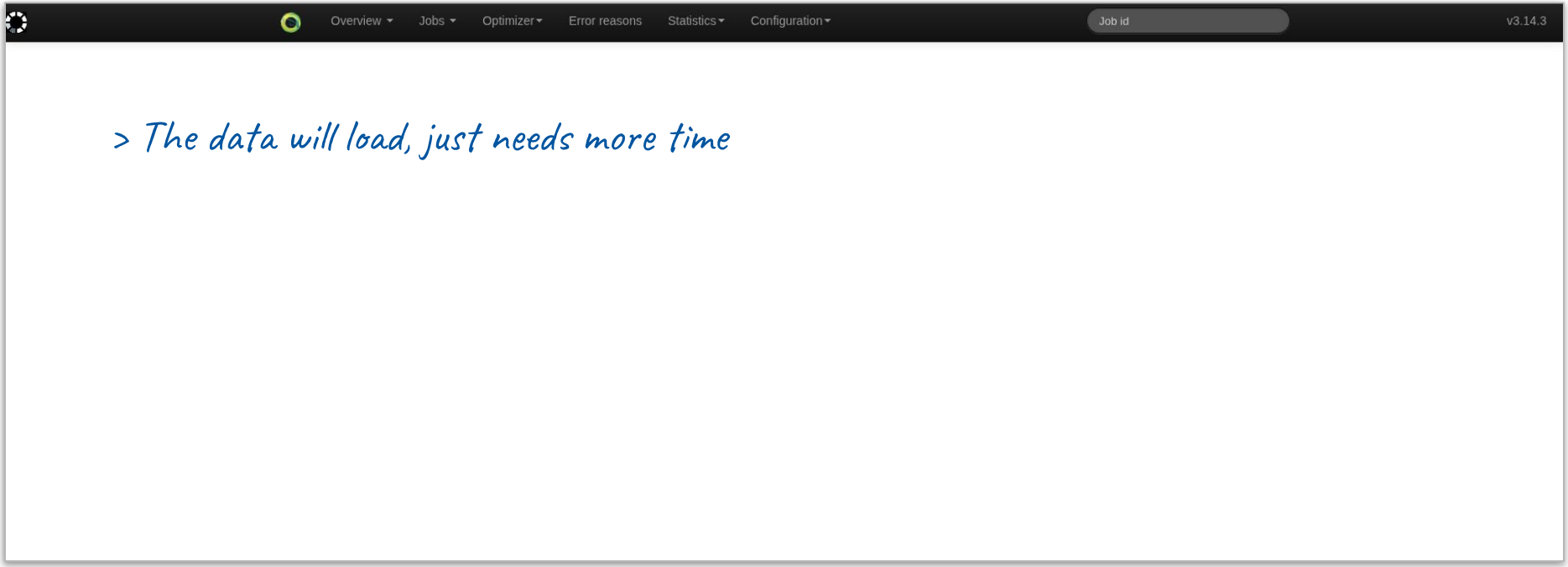
Special mention:
Amir Glick (FTS Summer Student 2026)

Problem definition

- Web Monitoring queries always stressed the system
- Database usage never optimized for time-based data
  (e.g.: `t_file` holding both active and terminal states)
- CERN FTS deploys primary + replica to compensate
  - Web Monitoring reads from the replica
- Even in this setup, the WebUI may still run slow under a heavy database



The notorious “Loading Overview” page



> The data will load, just needs more time

Attacking the problem

- Current system not well-suited to handle time-based data
- Why not try a time-series database?
- **TimescaleDB**: PostgreSQL extension offering time-series support

Amir Glick heroically spent his summer on integrating TimescaleDB with FTS !

Project plan

1. Evaluate TimescaleDB technology stand-alone
2. Evaluate TimescaleDB on heavy FTS data (i.e.: throughput data)
3. ???
4. TimescaleDB running on FTS4-Pilot

Step 1: TimescaleDB on its own

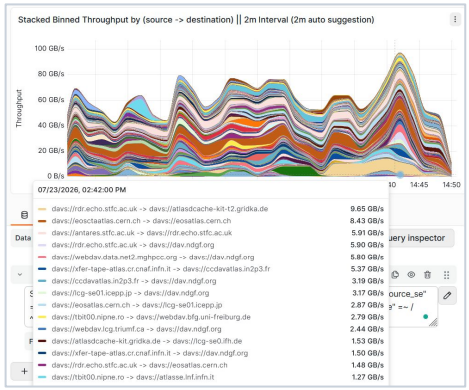
Simulate load: **10k links**, one throughput sample every **5 s**

	1 day of data	7 days of data
Rows	172 M	1.2 B
Ingestion	2.5 min	~15 min
Query 1-hour bins over 1-day of data	10 s	10 s
Query 1-hour bins over 7-day of data	–	1 min 21 s
Build materialized view: 1-hour bins	1 min 12 s	> 21 min
Materialized view: 1-hour bins over 1-day of data	0.55 s	0.56 s
Materialized view: 1-hour bins over 7-day of data	–	0.57 s

- Convincing numbers (TimescaleDB technology looks strong)
- Materialized views are a powerful technique

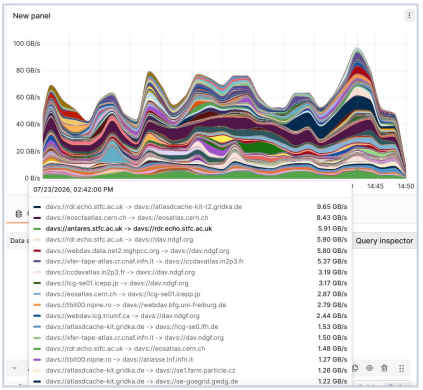
Step 2: Real FTS throughput data

- Existing FluentBit + InfluxDB pipeline for live-throughput data (only **internal** use, for debugging)
- Mirrored the existing FluentBit pipeline to feed into TimescaleDB as well
- Compared the InfluxDB vs TimescaleDB plots



InfluxDB

identical



TimescaleDB

- Data matches one-to-one & no performance discrepancy observed with TimescaleDB

Step 3: ???

- **Objective:** Add TimescaleDB to FTS
  - Aim for two types of data: throughput events & file-state counts
  - Must modify multiple systems: transfer-input processing, database file-state changes & WebUI monitoring queries
- **Constraint:** TimescaleDB integration cannot interfere with a production system
- **Wish:** Use existing production tests to produce data for TimescaleDB evaluation

Step 3: Add TimescaleDB to FTS codebase

Throughput

- Transfer progress updated in a single place in the codebase
- Leverage this for TimescaleDB throughput events
- Write directly to the TimescaleDB database

File states

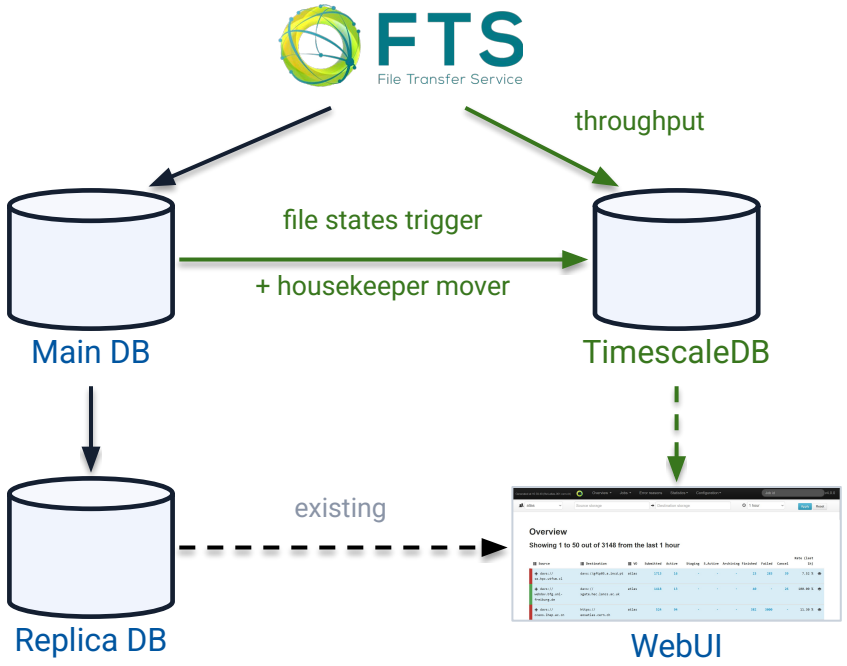
- State changes in many code paths
  - `getCancelJob`, `setRetryTransfer`, …

Triggers to the rescue!

- **Trigger** on `file_state`
  - Fills a short-lived table in the main DB
  - Housekeeping to move rows to TimescaleDB database

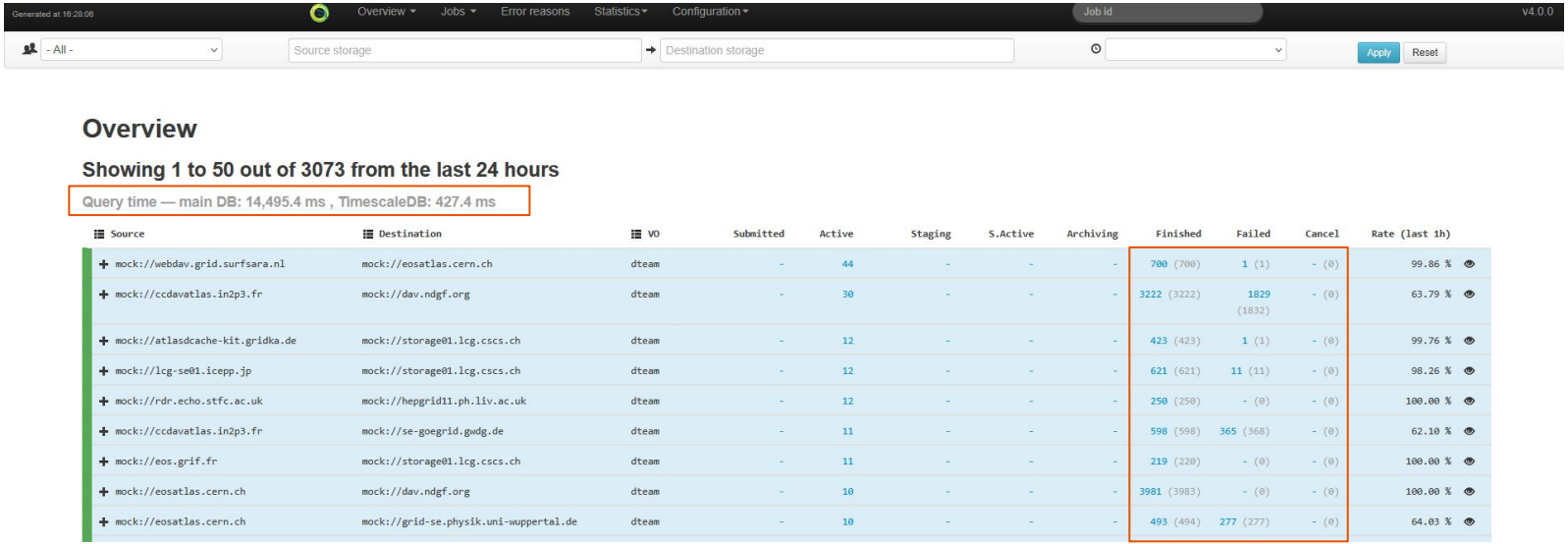
Step 3: A separate TimescaleDB

- “Separation of concerns” constraint
  - No extra load on the main database
  - Time-series data in its own database
- Separate database connection pool in FTS
- WebUI also queries the TimescaleDB database



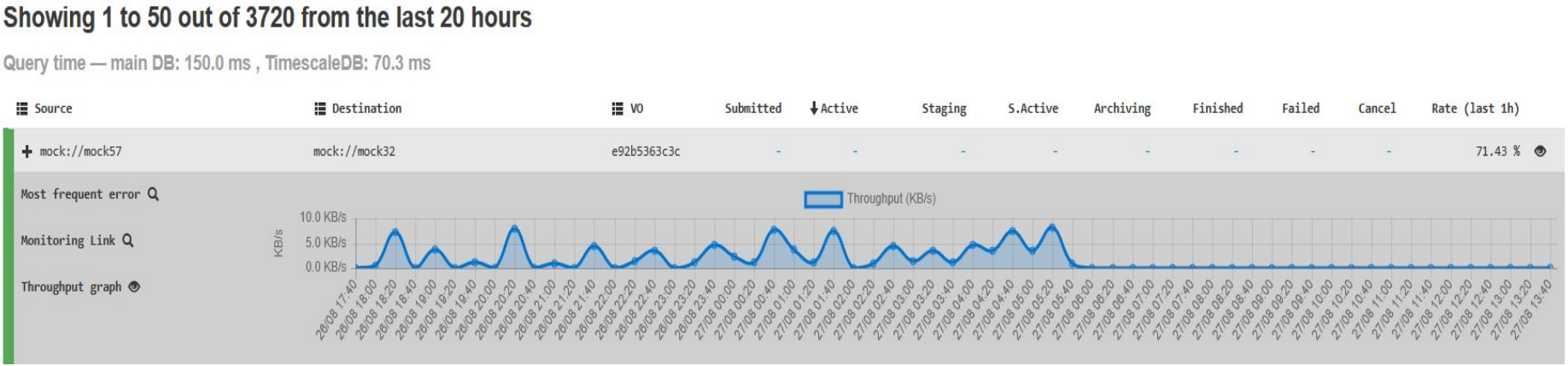
Step 4: FTS4-Pilot WebUI + TimescaleDB

- Overview queries served from TimescaleDB
- Query time of both databases reported side by side



Step 4: FTS4-Pilot WebUI + TimescaleDB

- **Bonus**: Throughput plots per `src → dst` link



Rite of Passage

FTS4-Pilot + TimescaleDB vs. ATLAS traffic patterns

Replaying FTS4-ATLAS traffic on FTS4-Pilot

- Reproduce FTS4-ATLAS traffic
  - Capture Monit-OpenSearch data from FTS4-ATLAS (finished transfers)
  - Submit to FTS4-Pilot as `mock://` transfers (same size, duration & final state)
- Cron job polling the FTS4-Pilot WebUI every 60s, 1-hour view (29 - 31st Aug 2026)
  - Log query time of Replica and TimescaleDB
- DB cache kept small on both databases `(shared_buffers = 100MB)`

Reproduce FTS4-ATLAS production submission & transfer patterns !

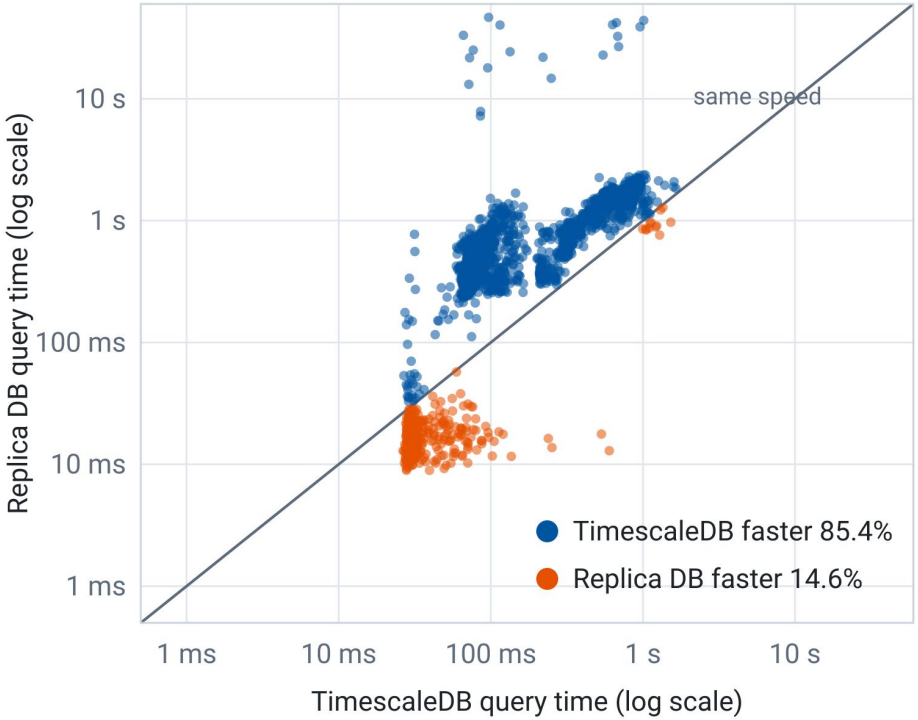
Results: Query time logs

2,487 polls of the FTS4-Pilot WebUI, 1-hour view

	Replica DB	TimescaleDB
Median	441.6 ms	84.8 ms
Mean	834.8 ms	236.3 ms
95th percentile	1.78 s	0.85 s
Slowest poll	46.7 s	1.6 s
Polls slower than 1 s	531	38

- TimescaleDB faster on **85.4%** of polls
- More importantly: **performance consistency** even during outliers

Results: Poll by poll



- Each dot: one WebUI poll (TimescaleDB vs Replica DB time)
  - Above the line: TimescaleDB faster
  - Below the line: Replica DB faster
- Replica DB wins mostly on very quick queries (we suspect data is entirely cached)

WebUI page on a "bad day":

Query time — main DB: 14,495.4 ms , TimescaleDB: 427.4 ms

Conclusion

- TimescaleDB fits seamlessly in the Postgres environment
- Very satisfied with the TimescaleDB performance measurements so far
- FTS team plans to adopt TimescaleDB
  - Work remains to drop the trigger and replace with codebase updates
  - Plans to include file state plots (finished, active, …)
- On the horizon → Move heavy WebUI queries to TimescaleDB
  → Opens the door to drop the monitoring replica

> *Thanks to Amir for sacrificing his summer!*

Thank you!