An Information Aggregation and Analytics System for ATLAS Frontier

J Lozano Bahilo[1], A Formica[2], E J Gallas[3], N Ozturk[4], M Si Amer[5], I Vukotic[6]

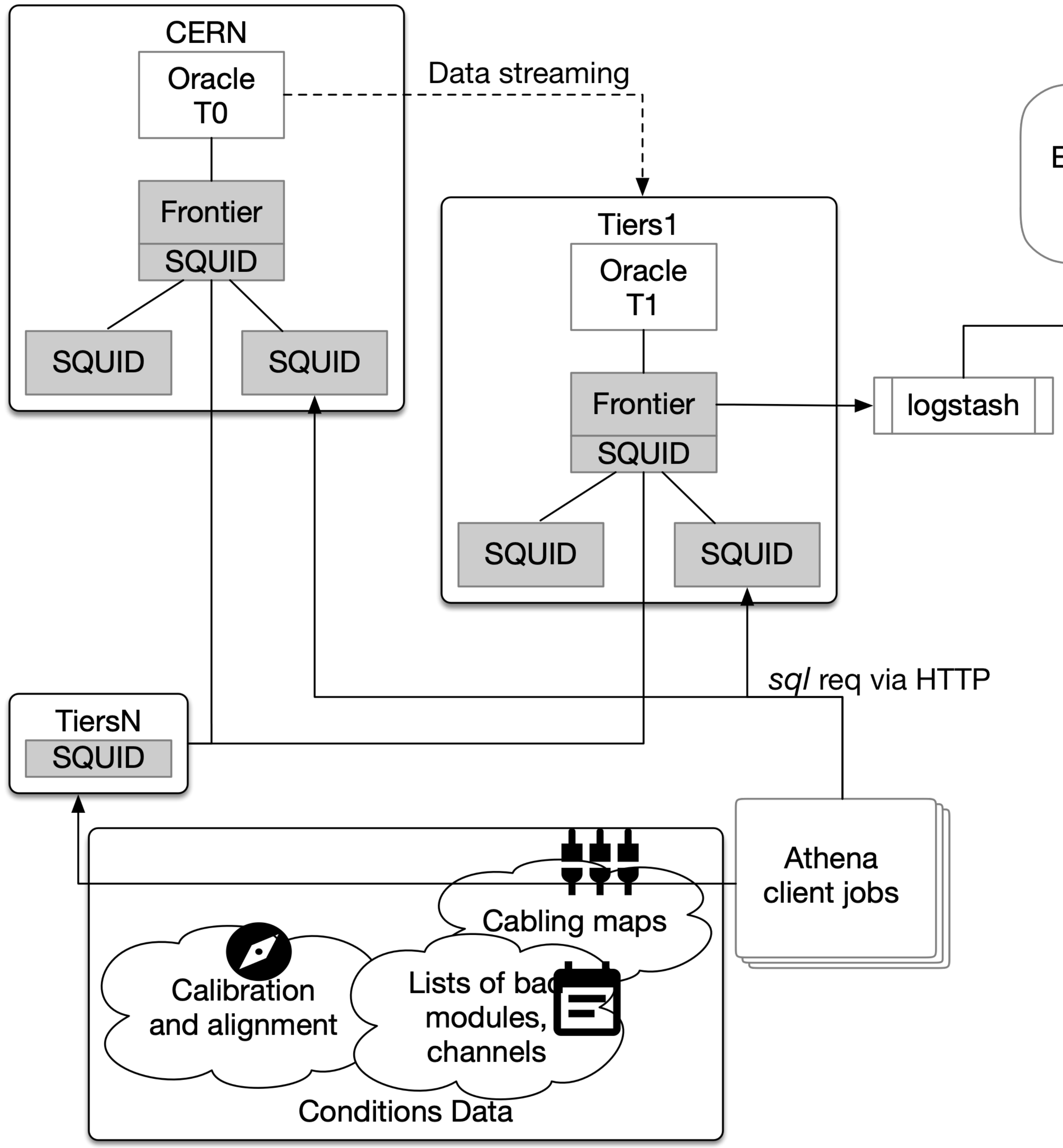
1. Univ. of Valencia and CSIC (ES)
2. Irfu - CEA Saclay (FR)
3. Department of Physics, University of Oxford (UK)
4. University of Texas at Arlington (US)
5. Ecole Nationale Supérieure d'Informatique (DZ)
6. University of Chicago (US)

Irfu - CEA Saclay
Institut de recherche sur les lois fondamentales de l'Univers

Outline

- Access to Conditions Data in ATLAS
- Motivations for an Analytics System
- Application Description
- Deployment
- Results

Access to Conditions Data in ATLAS



- ATLAS jobs access Conditions data using a distributed caching infrastructure based on Frontier-Squid system.
- Frontier server logs files are mined to index all Conditions data requests in an Elasticsearch repository in Chicago.
- Kibana dashboard is available to study requests and response metrics stored in Elasticsearch.

Motivations for an Analytics System

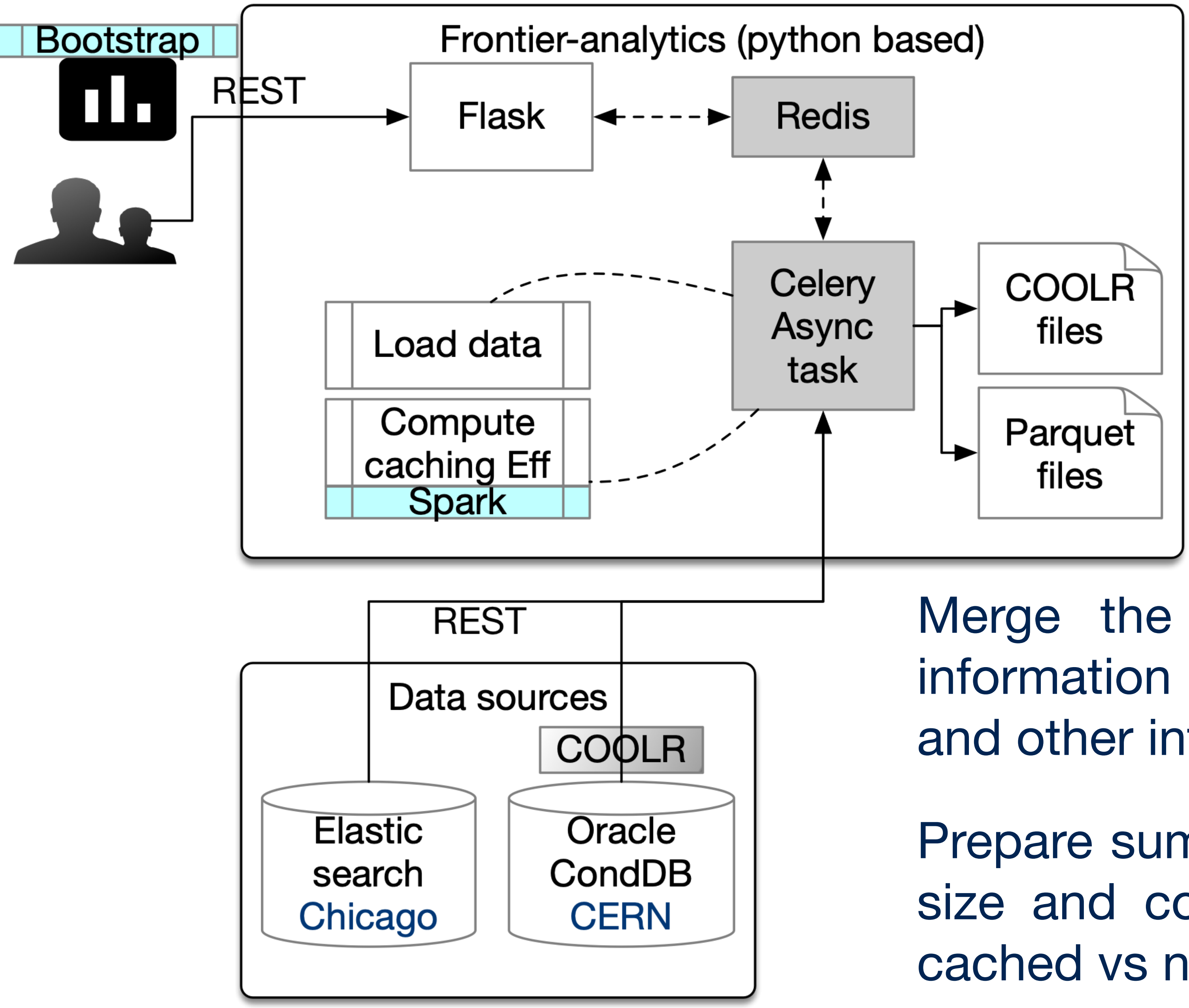
- The Frontier-Squid system suffered from service degradation leading to failures in particularly problematic workflows :
  - Real data underlying events "overlayed" on simulation data
  - Specialised reprocessing
- Requests from these workflows were much less likely to be found in the cache (i.e. low "cache efficiency").
- Using Frontier logs we could extract the SQL requests and re-play them on a separate Frontier instance or via COOLR services (a REST API to the ATLAS Conditions database COOL).
- The analysis found that several requests were accessing the same payload data but using different SQL requests (…different URLs => not cacheable).
- Identification of such problematic requests patterns is essential to improve the system for Run 3.

Project overview

- The Frontier-Analytics project has the purpose to process the requests of particular tasks or during particular time periods and derive summaries of key information which help to isolate the problematic requests for more detailed inspection.

- This application is based on python libraries and retrieve data from the existing infrastructure via REST APIs:
  - Log data information from Elasticsearch
  - Conditions payload and metadata from ATLAS COOL Conditions database.

Frontier Analytics Architecture



Select the task-id to explore via Flask.

Celery launches data loading from Elasticsearch. Output parquet files are produced for further processing.

Merge the Elasticsearch data with metadata information from COOLR (add tag, folder names and other information).

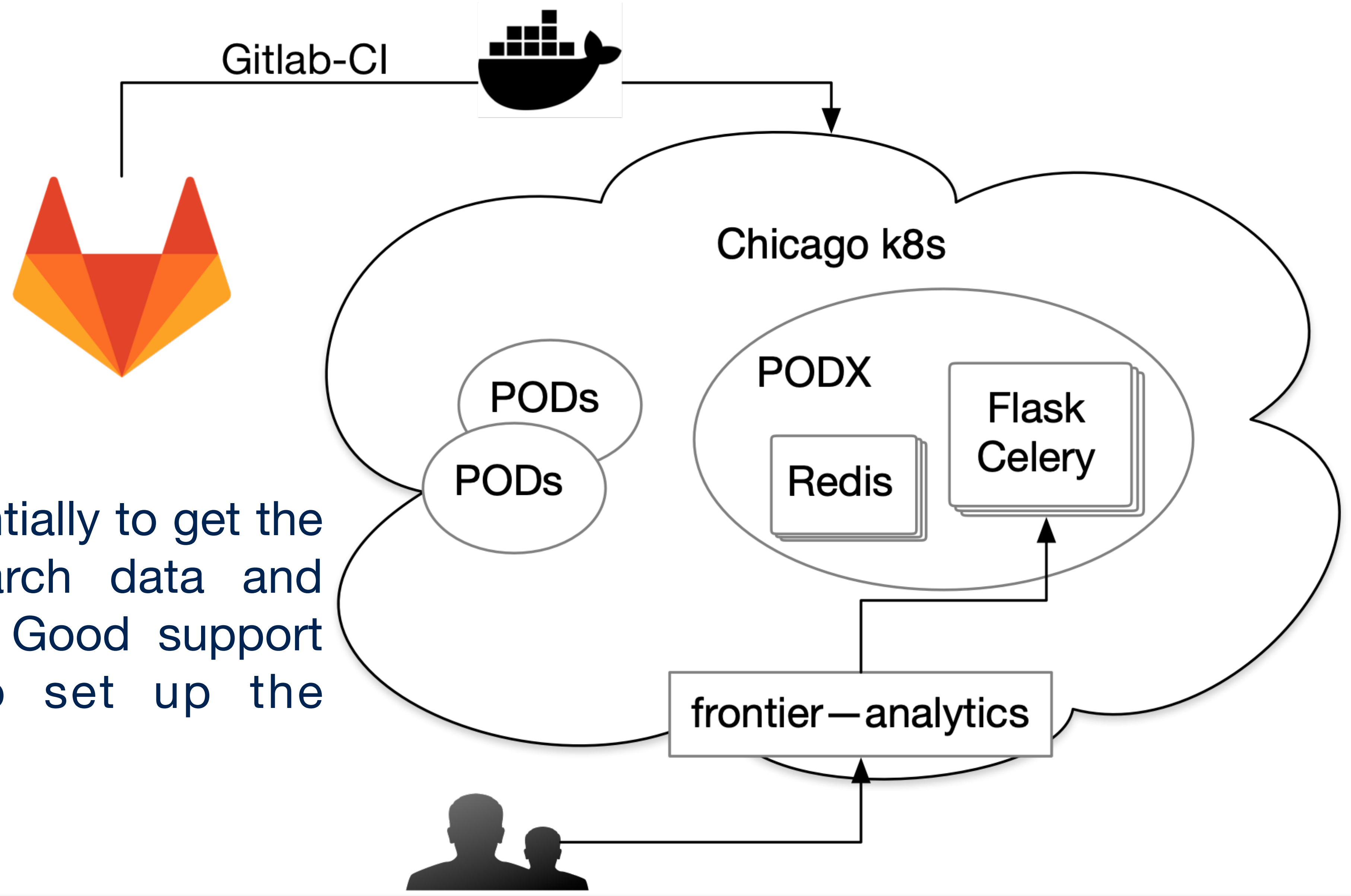
Prepare summary plots: requests' time, response size and counters (MC vs Data, # by schema, cached vs not-cached).

Compute caching efficiency for selected folders (payload data via COOLR).

Frontier Analytics Deployment

Deployment site: Chicago University

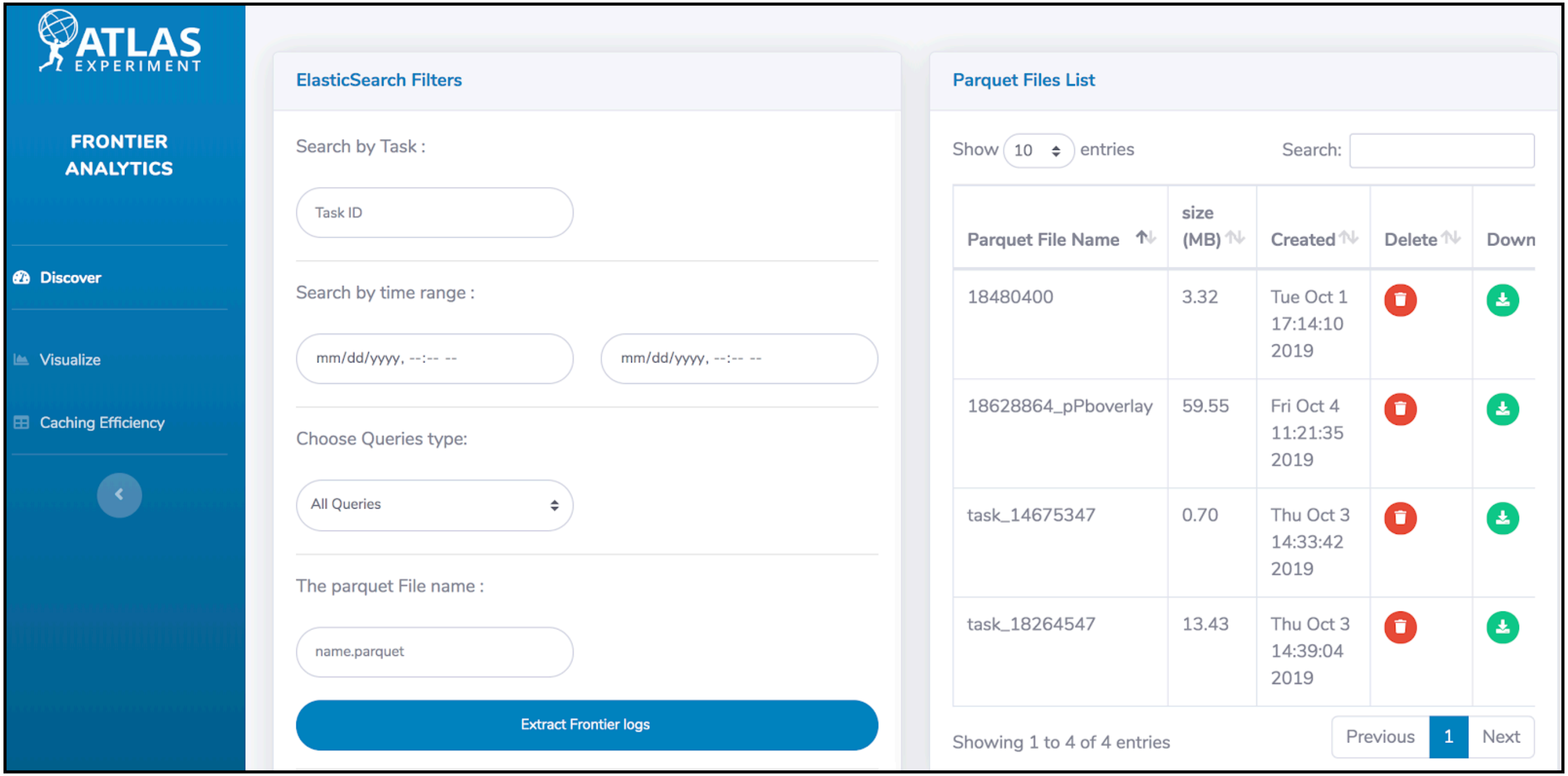
Usage of GitLab-CI process to prepare Docker images from CERN GitLab. The images can then be deployed in a Kubernetes cluster.



Choice of Chicago was essentially to get the system close to Elasticsearch data and improve the loading speed. Good support from Chicago experts to set up the deployment.

User Interface

https://frontier.uc.ssl-hep.org/



ElasticSearch Filters

Search by Task :

Task ID

Search by time range :

mm/dd/yyyy, --:-- --   mm/dd/yyyy, --:-- --

Choose Queries type:

All Queries

The parquet File name :

name.parquet

Extract Frontier logs

Parquet Files List

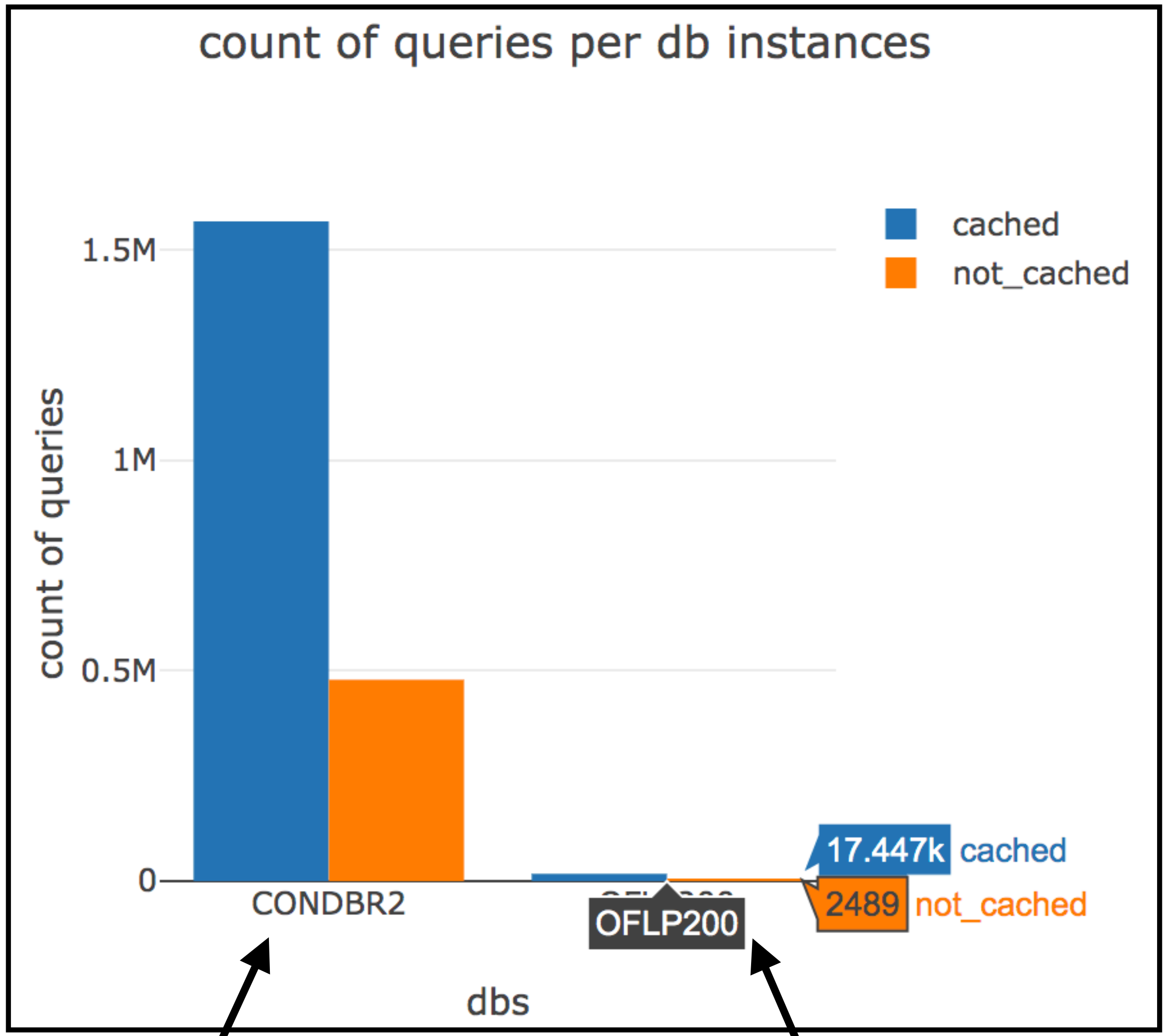
Show 10 entries   Search:

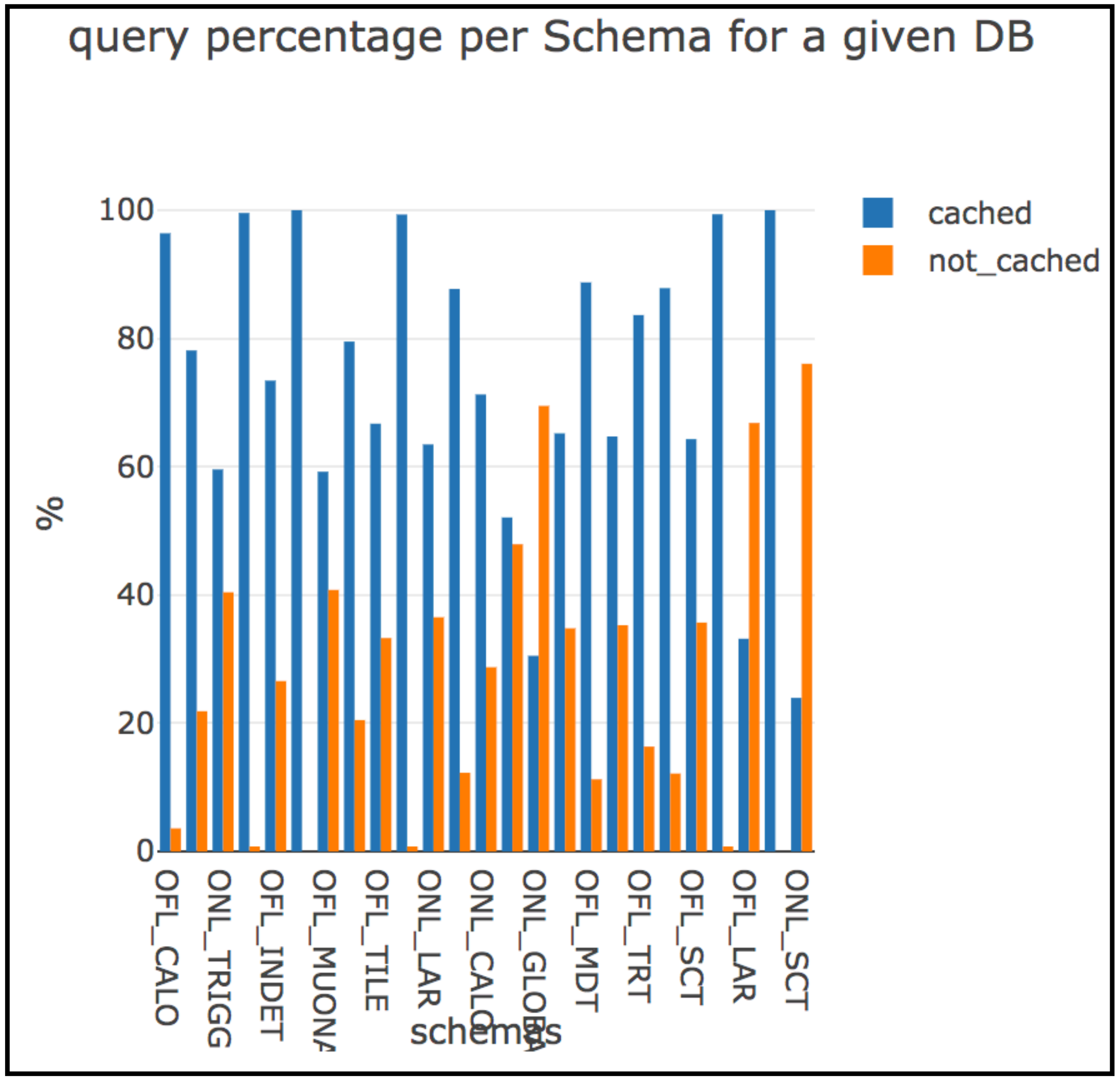
Parquet File Name	size (MB)	Created	Delete	Down
18480400	3.32	Tue Oct 1 17:14:10 2019		
18628864_pPboverlay	59.55	Fri Oct 4 11:21:35 2019		
task_14675347	0.70	Thu Oct 3 14:33:42 2019		
task_18264547	13.43	Thu Oct 3 14:39:04 2019		

Showing 1 to 4 of 4 entries   Previous 1 Next

Results: Visualize (1)

proton-lead overlay task: 18628864

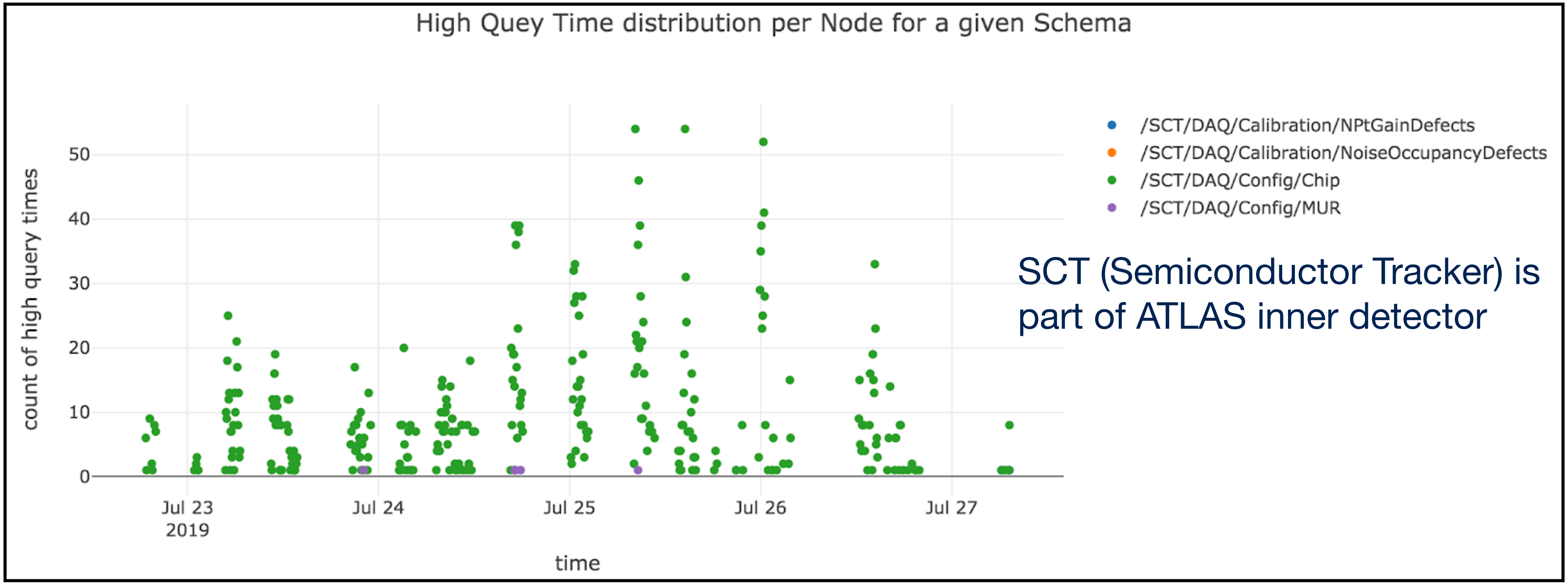




Data

MC

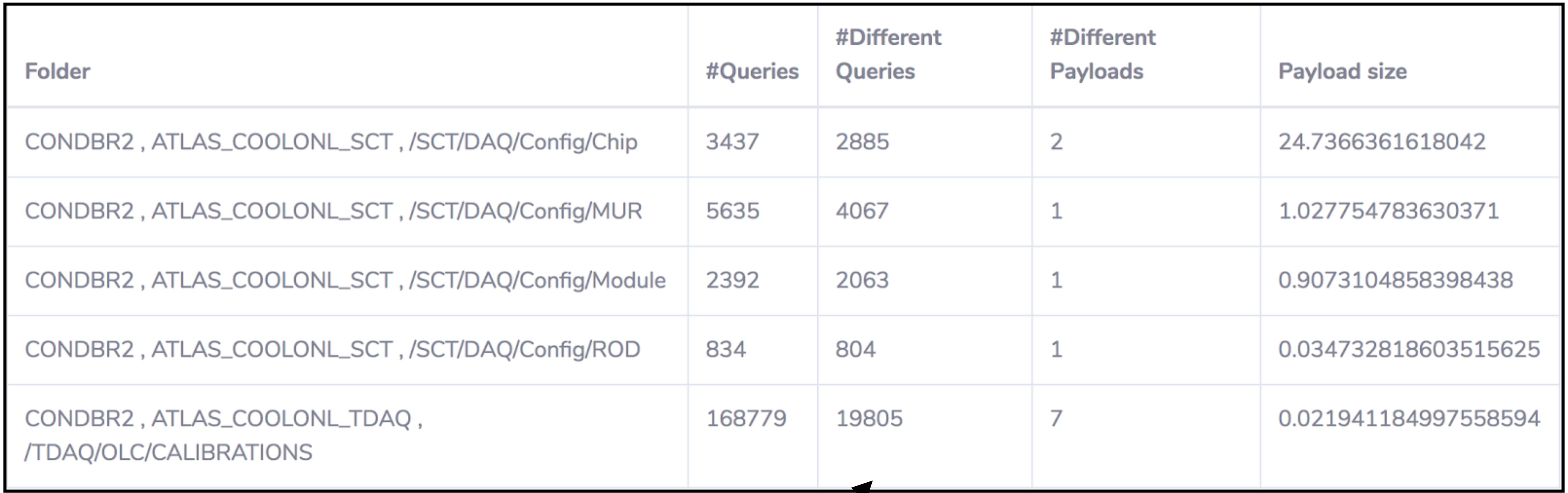
Results: Visualize (2)



SCT (Semiconductor Tracker) is part of ATLAS inner detector

Queries with high processing time (>1 s) are mostly from ONL_SCT, in particular from node /SCT/DAQ/Config/Chip

Results: Caching efficiency



Folder	#Queries	#Different Queries	#Different Payloads	Payload size
CONDBR2 , ATLAS_COOLONL_SCT , /SCT/DAQ/Config/Chip	3437	2885	2	24.736361618042
CONDBR2 , ATLAS_COOLONL_SCT , /SCT/DAQ/Config/MUR	5635	4067	1	1.02775478363037I
CONDBR2 , ATLAS_COOLONL_SCT , /SCT/DAQ/Config/Module	2392	2063	1	0.9073104858398438
CONDBR2 , ATLAS_COOLONL_SCT , /SCT/DAQ/Config/ROD	834	804	1	0.034732818603515625
CONDBR2 , ATLAS_COOLONL_TDAQ , /TDAQ/OLC/CALIBRATIONS	168779	19805	7	0.021941184997558594

The different queries are queries with a different range in time (so different SQL / URLs to Frontier-Squid).

The different payloads show the number of different conditions data retrieved by those queries.

Ideally we would like to have the same query for the same payload retrieved (to improve caching).

Size in MB for 1 payload

Summary of Problematic Workflows

Workflow type	Subsystems with high query time	Folders with bad caching efficiency
pPb, PbPb overlay: EVNT, DRAW->AOD	ONL_SCT, OFL_DCS, OFL_LAR OFL_TRIGGER, ONL_TDAQ	/SCT/DAQ/Config/Chip, /Module, /MUR, /ROD /TDAQ/OLC/CALIBRATIONS
Reprocessing: DRAW_RPVLL ->DAOD_RPVLL	ONL_SCT, OFL_DCS, ONL_TRIGGER ONL_LAR, ONL_RPC, OFL_TRIGGER	/SCT/DAQ/Config/Chip ONL_TRIGGER: /TRIGGER/Receivers/Conditions/VgaDac
Data scouting: calibration, DataScouting.merge.RAW -> AOD	ONL_TDAQ ONL_LAR	/TDAQ/OLC/LHC/FILLPARAMS /LAR/Configuration/FEBConfig/Physics/EMECC1
Perf-idtracking: pathena, DAOD_EGAM1->ROOT files	OFL_DCS (PIXEL and SCT) /PIXEL/DCS/HVCURENT	No calculation done: # of different queries will be equal to # of different payloads for DCS

Conclusions

We have an application to analyse problematic Conditions data access patterns in several workflows. This application benefits of the existing monitoring infrastructure in Frontier sites like CERN, CC-IN2P3 Lyon, TRIUMF, RAL and University of Chicago (thanks to all experts in each site!).

It will be used by experts in order to improve Conditions data access stability for current operations through LS2 as well as to design more cache-friendly system for Run 3.

The application is easy to deploy in a Kubernetes cluster, thanks to a complete Gitlab-CI chain in place.