

# Conditions evolution of an experiment in mid-life... ...without the crisis

*L. Rinaldi, A. Formica, E. Gallas, N. Ozturk, S. Roe*

*on behalf of the ATLAS Collaboration*

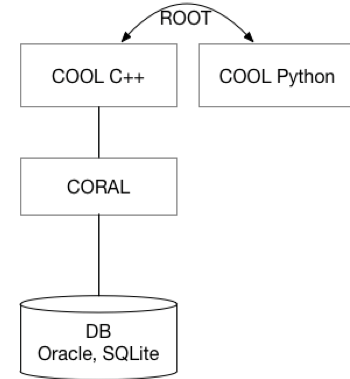


- The COOL experience: ATLAS Conditions DB during LHC Run-1 and Run-2
  - Description of the COOL/CORAL infrastructure
  - Conditions usage and COOL DB access
- CREST: the new Conditions REST DB infrastructure
  - Description of the infrastructure: the server and the client
  - Comparison of the CREST DB system with respect to COOL
  - Athena access to CREST
- The transition from COOL to CREST (during Run-3)
  - COOLR: a tool for COOL Rest access
  - Performance test: Gatling and Frontier like access
- Outlook and conclusions

Conditions data describe the state of ATLAS detector (very heterogeneous information)

During LHC Run-1 and Run-2, ATLAS Conditions data were managed by the [COOL/CORAL](#) system (by CERN-IT).

COOL is a C++ API based on CORAL client for access to the Relational (Oracle) DB instances (online, offline) and python binding is provided via [PyROOT](#)



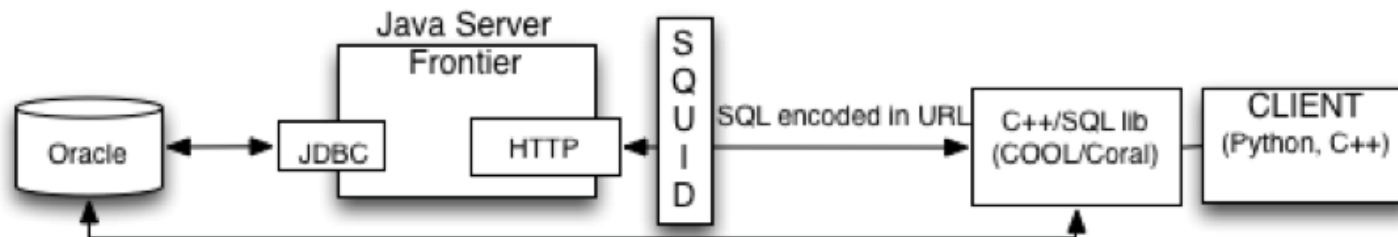
## COOL high level functionalities:

- Sub-systems have their own “COOL databases” (*Schemas*) and store payload data in dedicated tables (*Folders*)
- Payload data are stored according to Interval Of Validity (IOV)
  - Single-versioned Folders: IOVs can be appended (no overwrite, no *Tag* defined)
  - Multi-versioned Folders: IOVs can be stored in arbitrary way, Folder are Tagged (payload can be overwritten)
    - A Global Tag is a collection of multiple FolderTags
- Payloads are retrieved by providing IOV boundaries (and related Tag name)

Conditions data are concurrently accessed by large number of clients, i.e. jobs from Athena (the ATLAS software framework).

Access managed via COOL API using the intermediate [Squid/Frontier](#) services:

- DB queries (HTTP URL) created in the client code and sent to Frontier
- Access to the Oracle DB managed by Frontier via Java DB Connectivity (JDBC)
- PAYLOADs are cached for identical queries at the level of Squid Proxy



- *Caching*: Conditions data payload loaded at the same time as the IOV (some workflows are badly cached! )
- *DB structure*: Conditions data spread over 30 Schemas and 10k Folders (about 1TB per data taking period). Every system change corresponds to new set of Folders
- *Long Term Maintenance*: COOL API and CORAL will be supported by CERN IT until the start of LHC Run-3
- *Tags and Global Tags management*: delicate and time consuming activity
- *Data preservation*: use of sqlite files can become complex

A new architecture should be operational after LHC Run-3

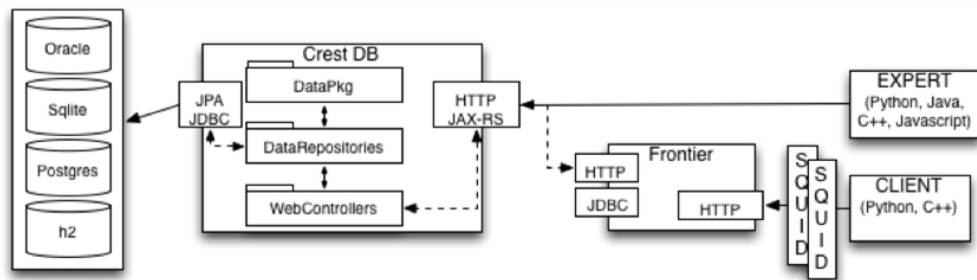
# CREST: a RESTful service for Conditions data

CREST is based on a Server-Client architecture, enhancing the role of the Frontier servers.  
The CREST development is a common activity with CMS (already in use by CMS during Run-2)

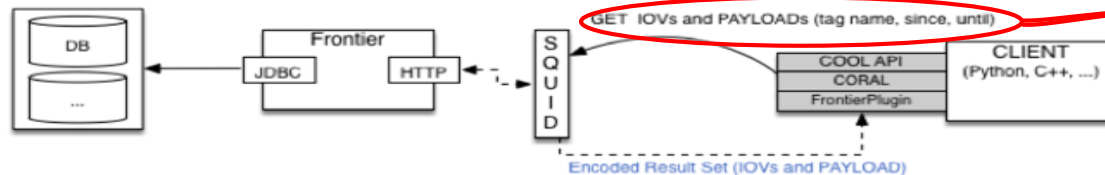
CREST DB server provides a full set functionalities to interact with backend DB (where Conditions data and metadata are stored):

- Functionalities exposed via Representational State Transfer (REST) architecture
- SQL not involved in server-client communication, internal resources accessible via URL and HTTP methods

Resource management is disentangled from the Client, allowing big simplification of the Client layer and giving more flexibility to backend implementation.



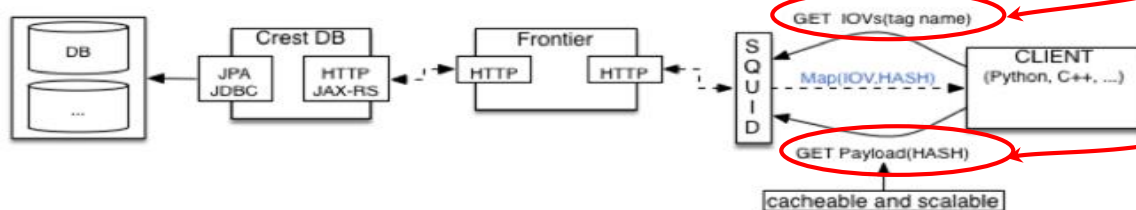
# CREST vs COOL: Client communication



**COOL:** SQL queries encoded in the URL containing IOV ranges parameter

- **retrieved response will contain IOV and PAYLOAD data**
- when cache is invalidated, the response is thrashed
- multiple HTTP calls needed

**The Big Change**



**CREST:** presence of the server interacting with DB backend and functionalities exposed via HTTP methods

- **PAYLOAD is disentangled from IOV access**
- even if IOV is invalidated, PAYLOAD is kept cached
- Payload requests never change (same url access same data on the backend): only 1 HTTP call is needed!
- Tag content information

Athena use of COOL is via a central service, **IOVDbSvc**

This has been modified to use HTTP requests accessing the CREST server, and data is provided in JSON format:

- Simple (single IOV) jobs run transparently (client/subsystem code unchanged) switching between COOL/CREST
- Can also produce or consume local data files in JSON format
- Now being extended to multiple IOVs



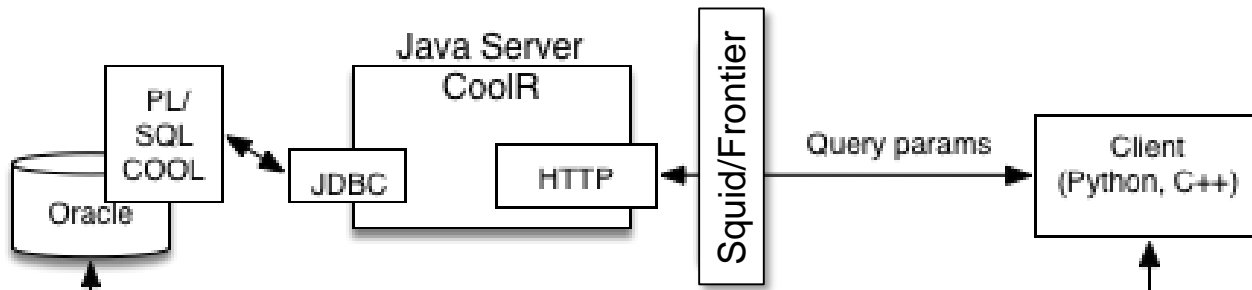
# COOLR: a tool for COOL REST access

Full CREST architecture will be **ready** by the **end of Run-3**

In the **transition** phase we need to preserve client COOL access in order to have a smoother migration, we are developing **COOLR** for a COOL REST access.

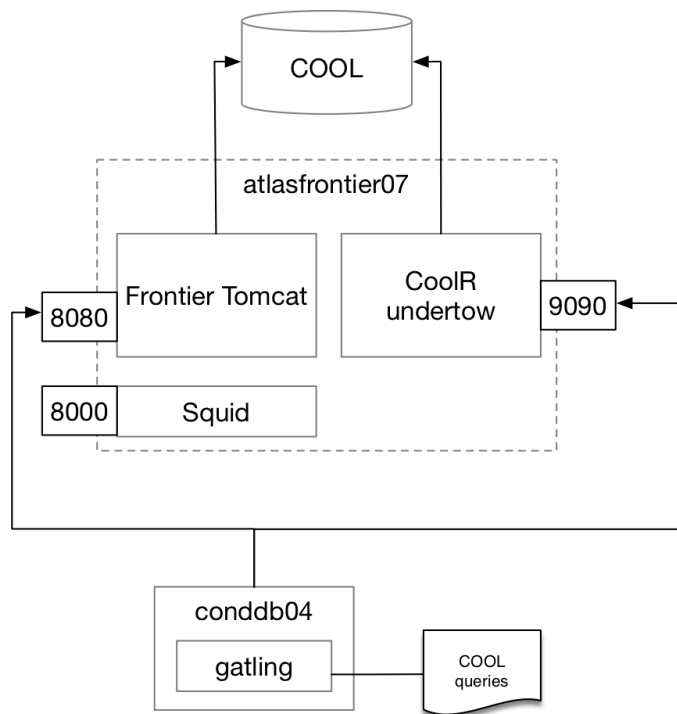
The COOLR server is accessed behind Frontier/Squid and can “read-only” COOL

The queries are stored in PL/SQL and, as in CREST, the URLs only contains query parameters: identical URLs can be cached by the Frontier/Squid with one HTTP call only!



The COOL-REST API has been specified using Swagger. Code has been generated to produce a REST API implemented using Java / Java API for RESTful Web Services

Testing machine enabled with an official Frontier launchpad installation.



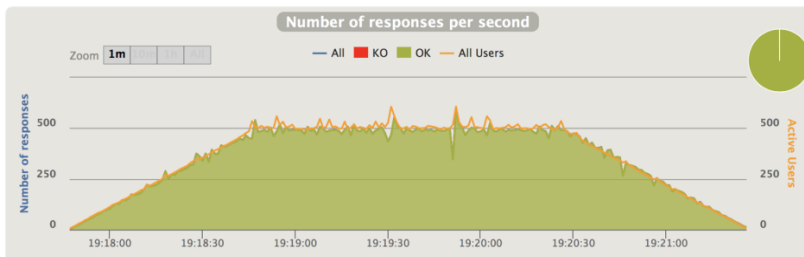
Frontier launchpad and COOLR server are listening on different ports.

Gatling framework and scripts (automatically generated via swagger) are deployed on another VM at CERN.

A set of “real” COOL queries was provided as input. 2000 queries are randomly taken by Gatling to simulate a load on the servers.

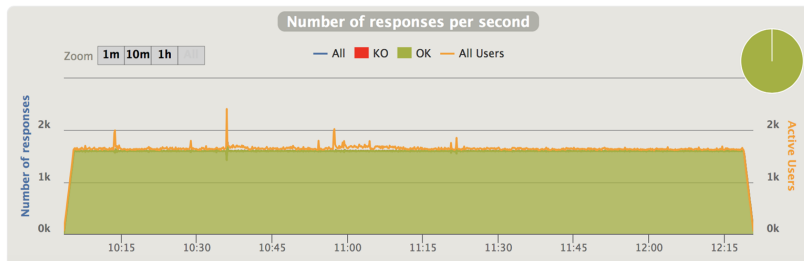
# Gatling testing COOL-REST

Aim: compare Frontier and COOLR server using the same input (encoded SQL).  
 This guarantees that we see a performance similar to the one of Frontier launchpad.



## Without Cache

Gatling test shows we can reach a pic point of 500 Req/s, without errors arising



## With Cache

Appending the SQUID to the launchpad we managed to reach > 1.5K Req/s (without errors).

This preliminary test shows that COOL REST access can sustain high request loads with respect standard Frontier access; more tests are in progress

- COOL DB has been used for storing Conditions data during LHC Run-1 and Run-2
  - The architecture is solid but it is not scalable with the future increase of ATLAS data processing jobs
- A new Conditions REST DB infrastructure (CREST) has been developed
  - The new system is based on a server-client architecture that allow a more simple and light management of Conditions payload and IOV data requests
- The transition from COOL to CREST is foreseen during LHC Run-3
  - Full CREST architecture at the end of LHC Run-3
  - Design IOVDbSvc for a REST access in Athena
  - COOLR: introducing REST technology into our existing DB framework
    - a prototype for COOL REST access is under evaluation
  - Preliminar Gatling test showed good performance of COOLR



# THANK YOU!

*lorenzo.rinaldi@unibo.it*



Istituto Nazionale di Fisica Nucleare



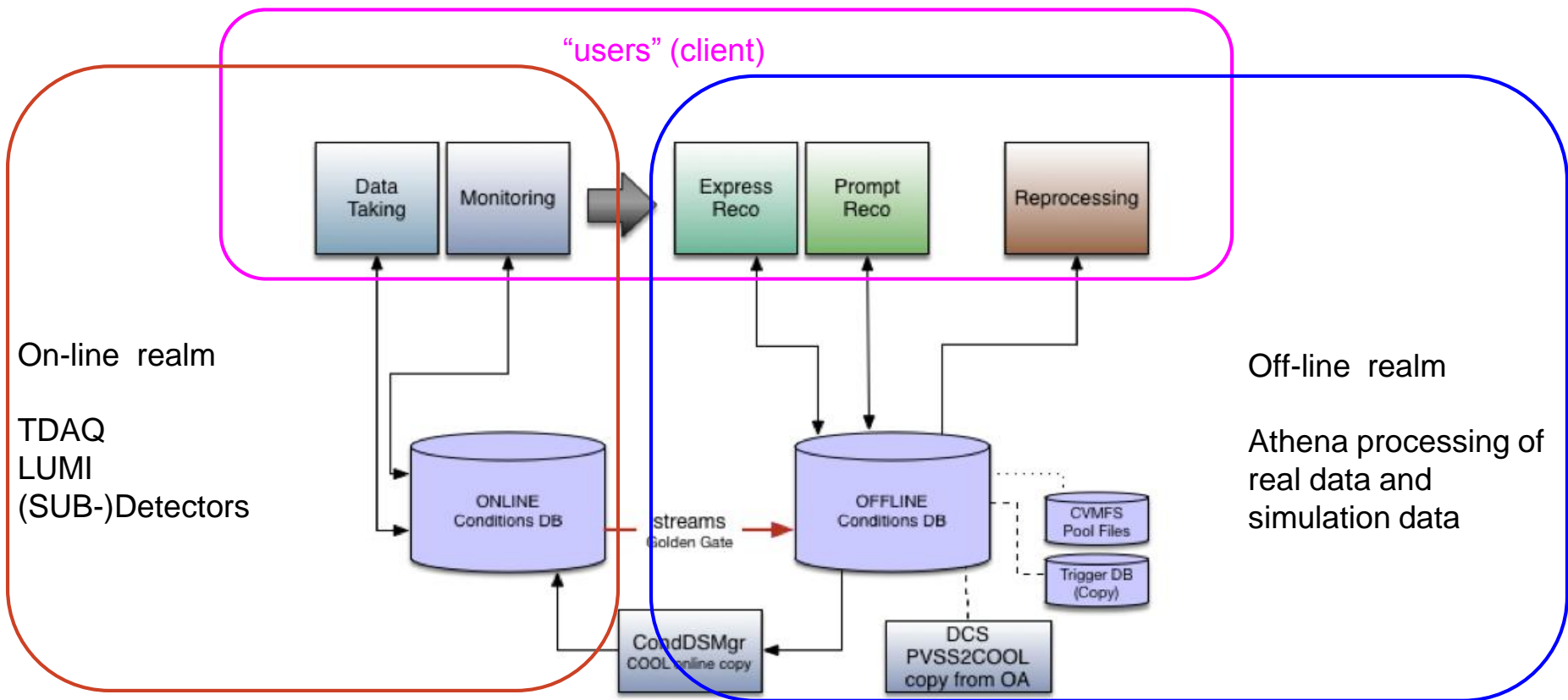
ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

*CHEP2018 - Sofia (Bulgaria)*

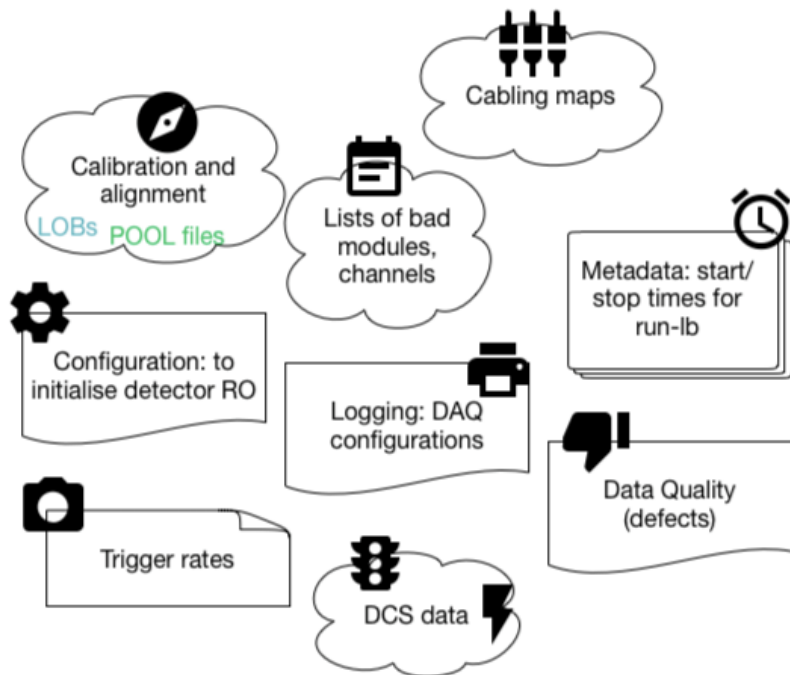
# Backup slides



# Conditions DB usage and data flow



# What is stored in the Conditions DB



Conditions data contain a large variety of information (such as Detector Control, trigger and DAQ, Data Quality, information from LHC and sub-detectors).

Heterogeneous information in:

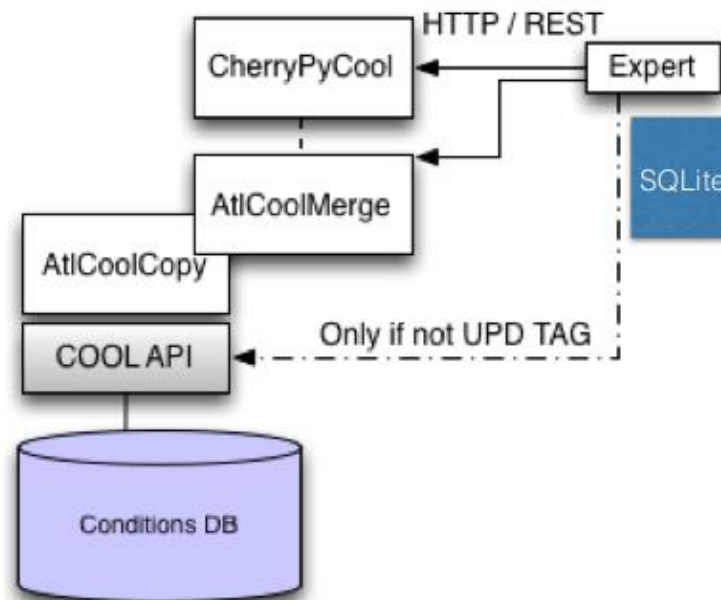
- data type (single column, Blobs, Clob, ext ref, POOL ROOT,...)
- granularity (from 1 IOV spanning 0-INF to sub-lumi level IOV)



# Writing data to COOL DB

Single-version Folders directly updated by expert

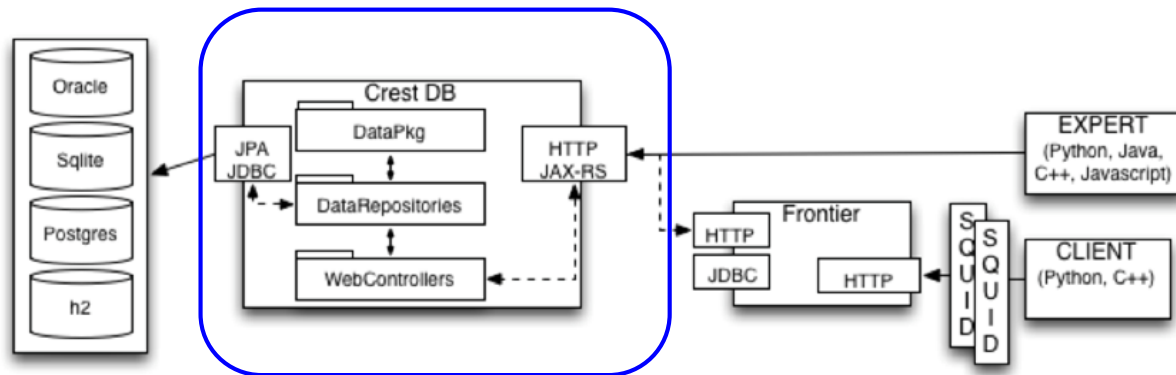
Multi-version Folders need special python tool for careful TAG handling (UPD TAG locking mechanism implemented)



# The CREST DB server

Crest DB SERVER: functionalities exposed via REST architecture

- SQL not involved in server-client communication, internal resources accessible via URL and HTTP methods
- Requests and Responses formatted in JSON
- Based on standard Java technologies (JEE, Spring) and specifications (JAX-RS, JPA)
- Can be deployed in the same Tomcat server as Frontier



# The CREST DB client

The client based on The Crest API (a set of REST methods), allowing external clients to interact with the resources represented in the data model.

Crest API is written using OpenAPI specifications

A JSON file describe the URLs and their parameters, and the Request and Response bodies

Clients and server stubs are generated via the the *swagger-codegen* library



One of the main goals is to improve the caching mechanism.

Using the information provided in the query parameters it is easy to decide caching policy based only on the expiration time (max-age parameter in the response header). Few examples:

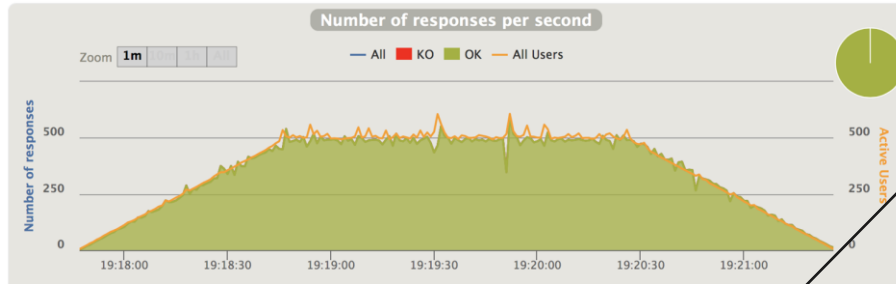
1. the query accesses a Tag: check if it is locked
2. the query accesses an IOV range: check if it is in the “past”

The Tag/IOVs insertion policies adopted in ATLAS and implemented via COOL do not allow for changes of conditions in the past for locked tags or for single version folders. This kind of query can thus be extended to large max-age values, without a check on update times.

# Gatling testing COOL-REST: no cache

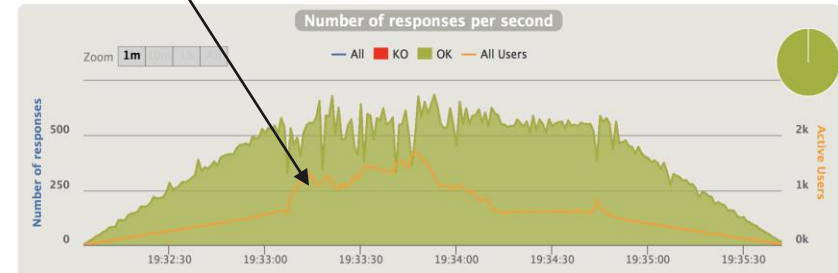
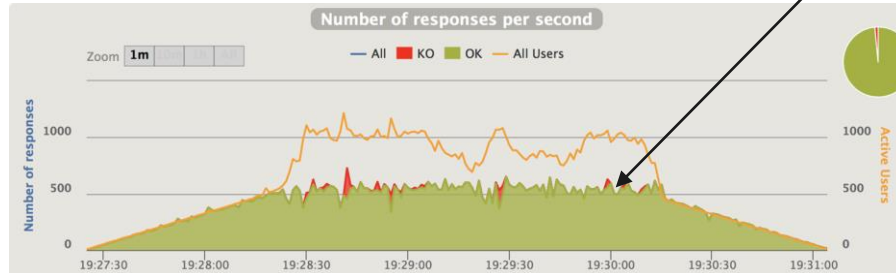
Compare Frontier and COOLR server using the same input (encoded SQL). This guarantees that we see a performance similar to the one of Frontier launchpad.

We show here a “working” example for Gatling tests (until about a pic point of 500 Req/s) and some other examples where we see the systems which are starting to suffer (in the pic range of 600 Req/s).



*Frontier (left):* server errors appear (config limit at 400 threads).

*CoolR (right):* number of active users increase because of queuing of Req.



# Gatling testing COOL-REST: via Squid

The SQUID of the launchpad was configured to redirect the requests containing in the URLs “CoolR-API” into port 9090. We managed to reach > 1.5K Req/s in this way (without errors).

