Information aggregation and analytics for ATLAS Frontier

OCTOBER 2018

AUTHOR:
Millissa Si Amer

SUPERVISORS:

Andrea Formica, Nurcan Ozturk, Torre Wenaus

CERN openlab

EXECUTIVE SUMMARY

CERN is a European Research Organization that operates the largest particle physics laboratory in the world. The Large Hadron Collider (LHC) at CERN has many detectors where the particles collide, the biggest detector called ATLAS generates a large amount of raw data per second, about 1 petabyte/second, and several Oracle databases are used to manage the processing of this data.

One of the most challenging ATLAS databases is the COOL database which stores the Conditions data in neutral schemas, it contains information about the calibrations, data-taking conditions and detector status.

More than 150 computing sites access this Conditions database using the squid-Frontier system which is a squid cache hierarchy used to cache the data at the computing site and make the access easier and faster, besides a Frontier web service to take charge of the different queries requiring data from the COOL database.

A decrease in the system performance has been noticed for some data processing workflows, that’s why the development of a monitoring and performance evaluation tool is necessary in order to study the system components in detail and detect possible anomalies. This report presents a CERN openlab project that consists of setting up an information aggregation and analytics tool in studying the breaking points of the squid-Frontier system.

ABSTRACT

Squid-Frontier system [1] is currently used to manage access to the COOL database [2]. This system includes many widely distributed computing sites and applications. Clients presented by PanDA (Production ANd Distributed Analysis system, the ATLAS' workload management system) applications are accessing this database to simulate and reconstruct physics events [3]. The goal of this distributed system is to cache the requested Conditions data and avoid direct access to the main Oracle database at CERN by using a cache proxy hierarchy and REST web services.

Different data processing workflows have been executed for the current LHC Run-2 data-taking at the grid computing sites, but some of them overloaded the Frontier service. The purpose of this project is to study the behavior of the squid-Frontier system and understand the bottlenecks of such workflows (overlay production in particular) on the grid. This study will help to improve the system performance and prepare for an adiabatic migration from the COOL data model to the CREST one.

Throughout this report, we will present the different tools and approaches used to study the squid-Frontier system and understand its operation. Moreover, we will explain the steps in setting up an information aggregation and analytics tool for extracting information from the squid and Frontier logs and analyzing them to detect the squid-Frontier system anomalies. Finally, the results will be shown by plots visualizing the relevant Frontier and COOL level parameters.

TABLE OF CONTENTS

1) INTRODUCTION

ATLAS data processing clients are accessing the ATLAS Conditions database via SQL queries that are processed by the squid-Frontier system at the computing sites. In order to ensure fast and efficient access for all clients, a squid cache hierarchy is set up between the clients and the Frontier web services. The purpose is to avoid overloading access to the COOL database by getting the data cached at the squids. This system works fine till some workflows (data overlay production) overload the Frontier servers and the database queries get disconnected or rejected which causes degradation of the squid-Frontier service. As a result the clients need months before completing the production process.

ATLAS team set up an infrastructure that allows them to extract logs from the squids and Frontier servers by using ElasticSearch technology. They have two servers :

- Rome ElasticSearch: for squid logs
- Chicago ElasticSearch: for Frontier logs

These servers offer Kibana as a monitoring tool. However, Kibana shows plots only the squid and Frontier level information, in order to analyze deeper the data and get more details about the SQL queries an CERN openlab project was suggested for developing an information aggregation and analytics tool for ATLAS Frontier system. In what follows, we explain in detail the different stages in setting up this tool and the development process.

2) SQUID- FRONTIER SYSTEM

For this project, we are interested in the squid-Frontier system at the Lyon computing site where a task from the overlay production was run in June 2018. There are two local squids and five Frontier servers set up at this site. For each Frontier server there is a local squid for caching Frontier data. This is a simplified view of the architecture in place:

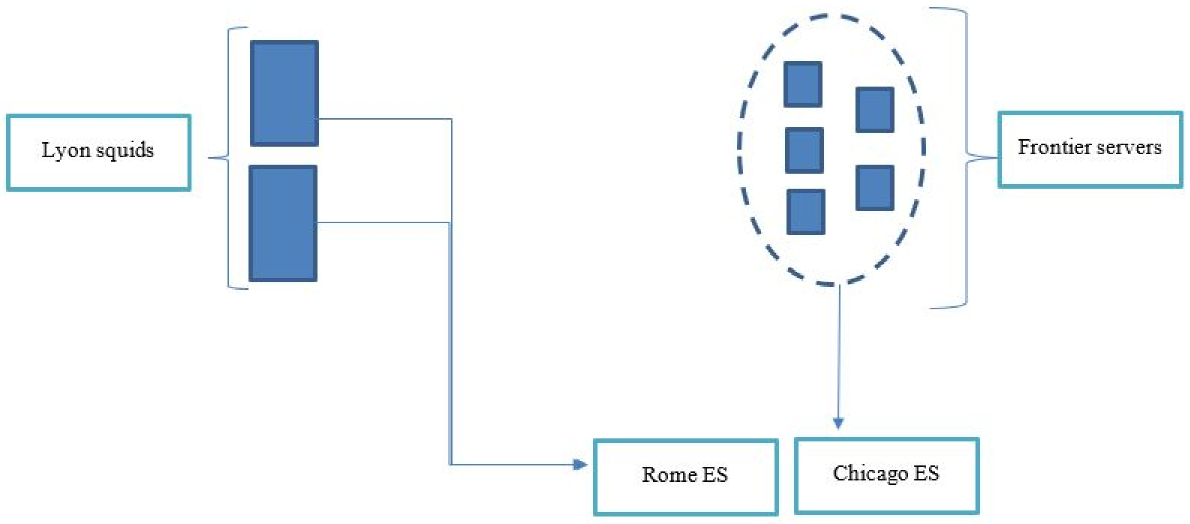


Figure 1. *The squid-Frontier system at Lyon site.*

When a client wants to get data from the COOL database, it sends a SQL query to one of the two local site squids. In case the data was cached already, the squid sends back the response to the client. Otherwise, the query goes to the Frontier server, if the data is not cached at the Frontier squid level then a request is generated and sent to the COOL Oracle database at Lyon. The response is cached at both the Frontier squid and the local site squid.

The squids checks the validity of the cached data every hour by sending requests to the Frontier servers asking if the data at COOL Oracle database has been modified or not. The Frontier checks from a table containing the different database schemas and their last update times whether the data cached at the squid level should be replaced or not.

a) THE COOL DATABASE

The COOL project provides tools for handling Conditions data for the LHC experiments. This project is the result of the collaboration between the CERN-IT department, ATLAS, CMS and LHCb experiments. The Conditions database is a huge relational database containing data about the calibrations, data-taking conditions and detector status.

The COOL database structure contains different nodes [4]. For a given database, the nodes have an id, a name and a path. Each node has a set of tables (schemas). A schema normally represents a detector sub-system specific data. Some tables have many versions. In this case, we differentiate between the versions using the tags. A tag is represented by an id (number) and a name (string), but usually we reference the tags in the queries by an usercond, an integer which represents the index of a tag in a table that contains the different tags for a given schema.

b) FRONTIER SERVER

The Frontier server is a REST web server using HTTP requests to communicate with the client. The Frontier service provides access to the Conditions, Trigger and Geometry databases [1]. The data is stored in the Oracle Offline Database ATLR at CERN (which is streamed from CERN to several Tier1 sites; RAL, TRIUMF, IN2P3). The purpose of the Frontier system is to reduce the load on the Oracle databases by:

1. Using squid caches to store the results of previous recent queries (many jobs request the same data)
2. Frontier launchpads are co-located with the Oracle servers, eliminating back/forth communication between the requestor and the database over the WAN.

c) SQUID

A proxy server is a server that acts as an intermediary between a client and a server. This proxy server can be used to cache data and improve the response time. Many proxy servers can be used in one cache hierarchy to increase the cache capacity and improve the Frontier system. In our case, between PanDA clients and Frontier servers we have a squid proxy hierarchy. A squid is caching and forwarding web server proxy that has a wide variety of uses, including speeding up a web server by caching repeated requests, caching web and DNS.

d) FRONTIER CLIENT

Frontier client is a C++ application that accesses the squids and the Frontier in order to access the Conditions database [1]. Clients run tasks, a task is a set of jobs that require data from the Oracle thus send queries to the Frontier system.

We are interested in studying the overlay task (with id = 14375483) that ran during 2 days (12-13th of June 2018). This is a typical task from the overlay production which creates stress on the Frontier servers. It overlays the data events from the 2016 proton-lead run onto the Monte-Carlo signal events. The task ran 140 reconstruction jobs (652 jobs upon retries) at the Lyon computing site IN2P3 where the local site squid monitoring in Kibana was set up. The squid cache-hit ratio is seen in Figure 2 below. A stress was seen at the site's Frontier server showing as disconnected queries (Frontier server got disconnected from the Conditions DB server) in the alert system, 25-50% of the queries (with squid code TCP_MISS) were not served by the local squid that went to the Frontier.

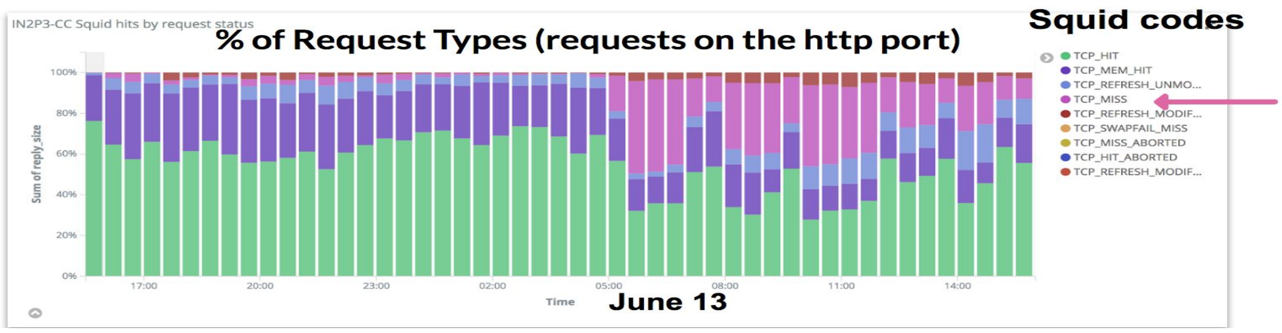


Figure 2. *Percentage of the Request Types (requests on the HTTP port) at the Lyon site squid.*

3) FRONTIER MONITORING

To study the behavior of the Frontier system, we have today an infrastructure which can gather a large amount of logging information from the Frontier launchpads and the squid servers from many sites [5]. As one can see in Figure 3 the log lines are read by using Filebeats and filtered by Logstash to aggregate information before storing them in ElasticSearch servers; one for Frontiers' logs at Chicago and another one for squids' logs at Rome. Here is a simplified view of the architecture in place:

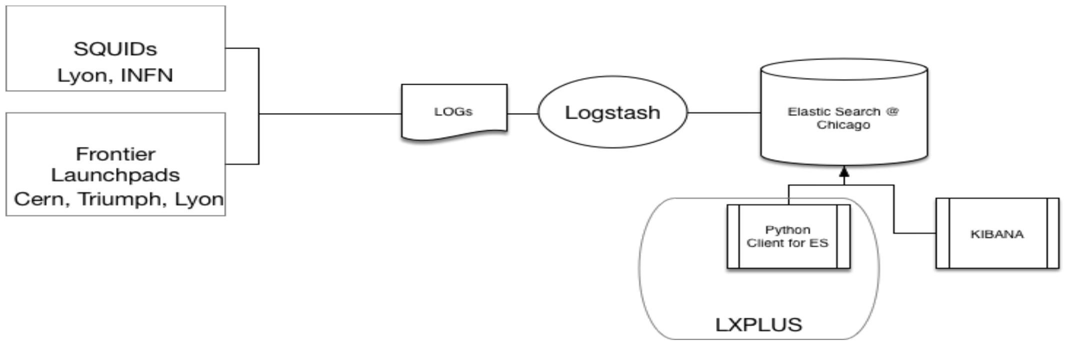


Figure 3. *Schema of the squid-Frontier monitoring system.*

CERN scientists could access to ElasticSearch servers by using LXPLUS clusters or Kibana dashboard.

a) ELASTICSEARCH

ElasticSearch provides a distributed, multitenant capable full-text search engine with an HTTP web interface and free JSON documents [6].

In the ElasticSearch server an index is like a “database” in a relational database, it has a mapping that defines multiple types. A type is a set of documents with properties, same as a Table in a relational database. Data is stored in documents which are similar to JSON documents for speeding up text searching.

b) LOGS CODE

To study log files, we gather all these logs information in ElasticSearch servers in different indices. An index as in a relational database is a place where to store the related documents. In our case, our Frontier logs are stored in the Frontier-new-* indices. An index in ElasticSearch can contain multiple types. These types can hold multiple documents. A document contains multiple variables as listed below [5]:

- TIME

- **@timestamp** (date): the time when the document was written

- QUERY AND QUERY STATUS

- **cached** (boolean): if true, data requested by the query is cached; no need to contact Conditions DB
- **dbtime** (number): time to get back the data from the Conditions DB (in msec)
- **disconn** (boolean): if true, the Frontier server got disconnected from Conditions DB server
- **finalthreads** (number): number of threads when the query is completed
- **fsize** (number): the total amount of data transferred by the Conditions DB server (in bytes)
- **initthreads** (number): number of threads when the processing of the query is initiated
- **locktime** (number): time to lock-up the connection to the Conditions DB (in msec)
- **nconnections** (number): number of connections established to the Conditions DB
- **nrows** (number): number of rows served by the Conditions DB
- **procerror** (boolean): if true, data got back by the Frontier server couldn't be processed (seems to be almost always of 0 sizes)
- **proctime** (number): time to processed the data returned by the Conditions DB until the query is completed (in msec)
- **queryid** (number): identifier of the query within the Frontier server (can be the same on different servers)
- **queryiov** (number): IOV of the SQL query request (in msec)
- **querysize**: number related to the length of the request
- **querytime**: total time spent since the query starts to be processed until it is completed
- **rejected** (boolean): query was rejected by Frontier server; seems that it is too busy to handle it (number of concurrent threads above the limit)

- HOSTS AND SERVERS

- **cllentmachine** (string): IP of the machine where the query originated
- **clientsquid** (string): squid on the client side
- **frontierserver** (string): Frontier server handling the query
- **squid** (string): squid on the server side

- USER

- **dn** (string): 'distinguished name'; related to the user information

- JOB AND TASK

- **pandaid** (number): Panda identification of the job
- **proctype** (string): type of the data processing done by the task
- **reqid** (number): request identifier of the task (complements the task identifier)
- **taskid** (string): Panda identifier of the task
- **taskname** (string): name of the task
- **tasktype** (string): type of the task
- **transpath** (string): transformation path of the task

- SQL QUERY

- **sqlhash** (number): checksum made with the SQL query to identify identical queries
- **sqllength** (number): length of the SQL query
- **sqlquery** (string): full SQL query
- **tableowner** (string): owner of the table being queried if it is present in the SQL query

The squid logs are stored in a different ElasticSearch server at Rome, in the Squids-* indices. The squid documents have other variables that express the squid's query status :

- **TCP_HIT:** A valid copy of the requested object was in the cache.
- **TCP_MISS:** The requested object was not in the cache.
- **TCP_REFRESH_HIT:** The requested object was cached but *STALE*. The IMS query for the object resulted in "304 not modified".
- **TCP_REF_FAIL_HIT:** The requested object was cached but *STALE*. The IMS query failed and the stale object was delivered.
- **TCP_REFRESH_MISS:** The requested object was cached but *STALE*. The IMS query returned the new content.
- **TCP_CLIENT_REFRESH_MISS:** The client issued a "no-cache" pragma, or some analogous cache control command along with the request. Thus, the cache has to prefetch the object.
- **TCP_IMS_HIT**: The client issued an IMS request for an object which was in the cache and fresh.
- **TCP_SWAPFAIL_MISS:** The object was believed to be in the cache, but could not be accessed.
- **TCP_NEGATIVE_HIT:** Request for a negatively cached object, e.g. "404 not found", for which the cache believes to know that it is inaccessible. Also, refer to the explanations for *negative_ttl* in your *squid.conf* file.
- **TCP_MEM_HIT:** A valid copy of the requested object was in the cache *and* it was in memory, thus avoiding disk accesses.
- **TCP_DENIED:** Access was denied for this request.
- **TCP_OFFLINE_HIT:** The requested object was retrieved from the cache during offline mode. The offline mode never validates any object, see *offline_mode* in *squid.conf* file.
- **UDP_HIT:** A valid copy of the requested object was in the cache.
- **UDP_MISS:** The requested object is not in this cache.
- **UDP_DENIED:** Access was denied for this request.
- **UDP_INVALID:** An invalid request was received.
- **UDP_MISS_NOFETCH:** During "-Y" startup, or during frequent failures, a cache in hit the only mode will return either UDP_HIT or this code. Neighbors will thus only fetch hits.

There are other squid codes related to the client side and hierarchy codes. For our study, the squid codes TCP_HIT and TCP_MISS are the most important.

c) KIBANA

Kibana is an open source analytics and visualization platform that allows interacting and visualizing the data in a variety of charts, plots, and maps. Kibana dashboard for the Chicago server shows all plots and charts for the Frontier data. It ensures a real-time monitoring and sends alerts in case of overloads.

A user can edit a query and filter ElasticSearch data to get only the data related to a specific production task or even a job. It can display plots, count rows or apply any aggregate function on a set of input data.

4) SOLUTION

For the overlay stress task (with Id = 14375483) at the Lyon site, the local squid Kibana monitoring showed between 25-50% of the queries (with squid code TCP_MISS) were not served by the local squid that went to the Frontier. Kibana detects also a set of disconnected queries; the queries not cached at the squid level and not connected to the Oracle database, which are retried many times in order to get the data. Due to these not cached queries, the quality of the service is degrading and the system is overloaded.

Kibana shows only plots related to Frontier parameters as the number of threads, response time, etc. From these plots ATLAS scientists can’t figure out the real issue in the squid-Frontier system at Lyon, that’s why a system which is capable of using the monitoring information in order to better understand the breaking points of the Frontier system, spotting possible weaknesses in the workflows or even detecting issues in the data model of the COOL database itself provides the best tool to understand and resolve the squid-Frontier problem.

Here are the different steps to set up an aggregation and analytics service for squid-Frontier system:

a) EXTRACTING DATA

The first step in the development process is to extract logs information from the ElasticSearch servers. The Figure 4 presents the different steps for the extraction process:

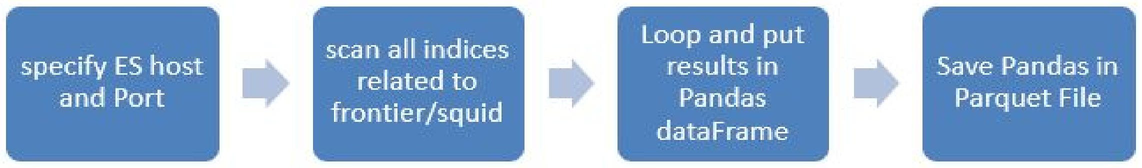

specify ES host and Port
scan all indices related to frontier/squid
Loop and put results in Pandas dataFrame
Save Pandas in Parquet File

Figure 4. *Data extraction process from ElasticSearch.*

Python3 was the main programming language used for the extraction of the ElasticSearch data, these are the different environments used with Python3:

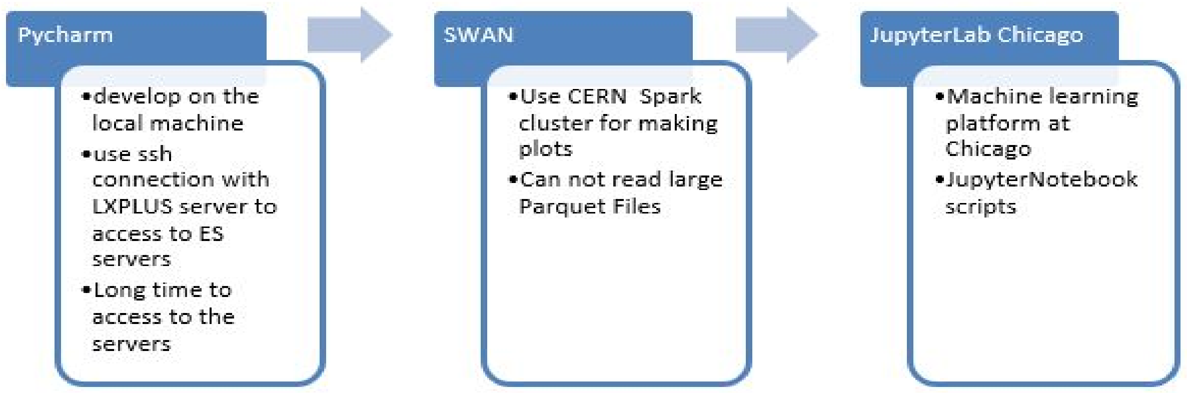

Pycharm
•develop on the local machine
•use ssh connection with LXPLUS server to access to ES servers
•Long time to access to the servers
SWAN
•Use CERN Spark cluster for making plots
•Can not read large Parquet Files
JupyterLab Chicago
•Machine learning platform at Chicago
•JupyterNotebook scripts

Figure 5. *Development tools.*

We are using the JupyterLab platform in Chicago as our development environment because :

- Simple environment, with Jupyter notebook scripts
- It accesses quickly to ElasticSearch servers, especially at Chicago
- Spark SQL can be used in JupyterLab

After getting data from both Chicago and Rome servers, we need to check data validity for the overlay task by comparing our results with Kibana. For the Lyon overlay task, we were interested more in studying the Frontier logs, here are the different counts of the queries from the Chicago ElasticSearch server:

Cached	Not-cached	Disconnected	Unique Disconnected	Processing error	Rejected	Total queries
606518	326214	740	709	0	0	932732

b) PARSING QUERIES

The second step in the development process is to analyze data by parsing the SQL queries. Parsing a query allows us to get all the COOL level parameters that are present in the query string.

Let’s study an example where we extract database fields from the SQL string of a disconnected query;

SQL QUERY	Database fields
'SELECT /*+ NO_BIND_AWARE QB_NAME(MAIN) INDEX(@MAIN COOL_I3@MAIN (CHANNEL_ID IOV_SINCE IOV_UNTIL)) LEADING(@MAIN COOL_C2@MAIN COOL_I3@MAIN) USE_NL(@MAIN	this query asks for these attributes which are columns in a SQL view : ● Object Id ● Channel Id ● IOV_SINCE

COOL_I3@MAIN) INDEX(@MAX1 COOL_I1@MAX1 (CHANNEL_ID IOV_SINCE IOV_UNTIL)) */ COOL_I3.OBJECT_ID AS "OBJECT_ID", COOL_I3.CHANNEL_ID AS "CHANNEL_ID", COOL_I3.IOV_SINCE AS "IOV_SINCE", COOL_I3.IOV_UNTIL AS "IOV_UNTIL", COOL_I3.USER_TAG_ID AS "USER_TAG_ID", COOL_I3.SYS_INSTIME AS "SYS_INSTIME", COOL_I3.LASTMOD_DATE AS "LASTMOD_DATE", COOL_I3.ORIGINAL_ID AS "ORIGINAL_ID", COOL_I3.NEW_HEAD_ID AS "NEW_HEAD_ID", COOL_I3.R_VMEAS AS "R_VMEAS", COOL_I3.R_IMEAS AS "R_IMEAS", COOL_I3.R_STAT AS "R_STAT"	● IOV_UNTIL ● USER_TAG_ID ● SYS_INSTIME ● LASTMOD_DATE ● ORIGINAL_ID ● NEW_HEAD_ID ● R_VMEAS ● R_IMEAS ● R_STAT
FROM "ATLAS_COOLOFL_DCS"."CONDBR2_F0020_CHANNELS" "COOL_C2", "ATLAS_COOLOFL_DCS"."CONDBR2_F0020_IOVS" "COOL_I3"	The attributes belonging to OFL_DCS schema located in a node with id = 20
WHERE COOL_I3.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_I3.IOV_SINCE>=COALESCE((SELECT /*+ QB_NAME(MAX1) */ MAX(COOL_I1.IOV_SINCE) FROM ATLAS_COOLOFL_DCS.CONDBR2_F0020_IOVS COOL_I1 WHERE COOL_I1.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_I1.IOV_SINCE<=1466917233318664161),1466917233318664161) AND COOL_I3.IOV_SINCE<=1466917333318664161 AND COOL_I3.IOV_UNTIL>1466917233318664161 ORDER BY COOL_I3.CHANNEL_ID ASC, COOL_I3.IOV_SINCE ASC'	where statement in SQL is used to select and filter a set of table rows. In this case, the where statement specifies an IOV range of the rows that should be returned from OFL_DCS

For our study, we selected a set of the parsed parameters that will be used in order to detect if the breaking point of the system is in the database organization [4]:

- **db** :COOL instance name.
- **schema**: Full COOL schema owner, the string contains a designation given to schemas used “online” (COOLONL) or “offline” (COOLOFL) plus ATLAS subsystem short name in COOL (DCS, TRT, etc).
- **node id**: unique integer for a folder of an account (schema) and instance.
- **node name**: folder set or folder name.
- **node full path**: full path of the node name.
- **tag name**: folder tag name (version).

- **tag id**: integer for a version of a folder.
- **node type**: indicate if the run express time or rows number for a tag.
- **run since:** the start of a run time converted to Oracle time format.
- **run until**: the end of a run time converted to Oracle time format.
- **lb (since,until):** the start and end of the first LB run.

To parse the queries, we used the COOLR service developed by Andrea Formica which is a REST web service providing a simple API to access to the COOL database. The usage of this service allows to more easily deploy the client code on the platform we did use for analytics because there is no dependency on the COOL API. For our purpose, we did install a simple Python client library. All the parsed data is saved in the parquet files and used for analytics purposes.

c) ANALYTICS

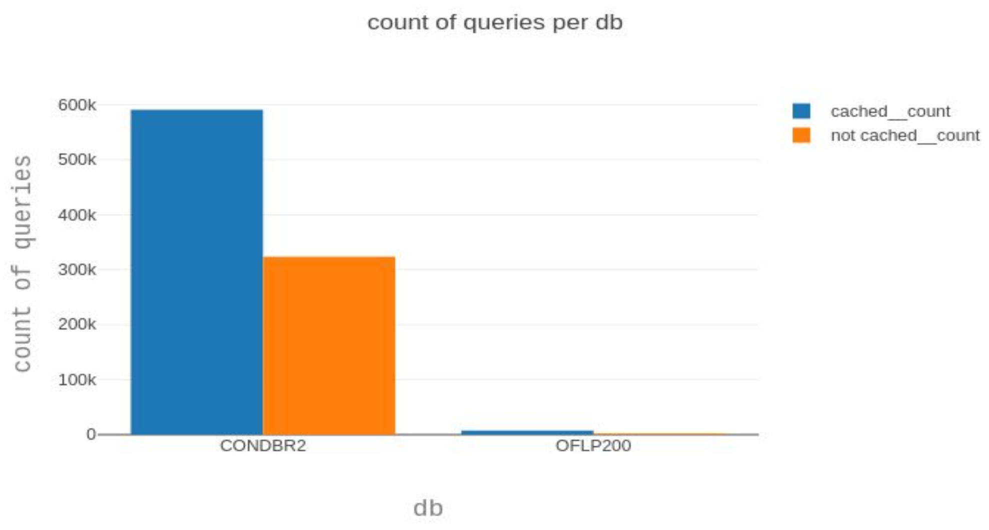
Once the data extraction is operational, the next step is to analyze the squid-Frontier performance by visualizing the data and suggesting a set of approaches and tools for performing analytics on the data.

A python script is developed to make plots from the parquet files using the Plotly library. This script offers a group of plotting functions where one can draw :

- a simple (x,y) plot
- a multiple y values plot
- a multiple (x,y) couples plots depending on a key

Besides the script offers two plot types; a line or a bar chart. Using the parquet files, different kind of plots were made to visualize the data and search for dangerous “query” patterns which could be associated with the degradation of our caching system.

1. Count of the queries per COOL instance (db):

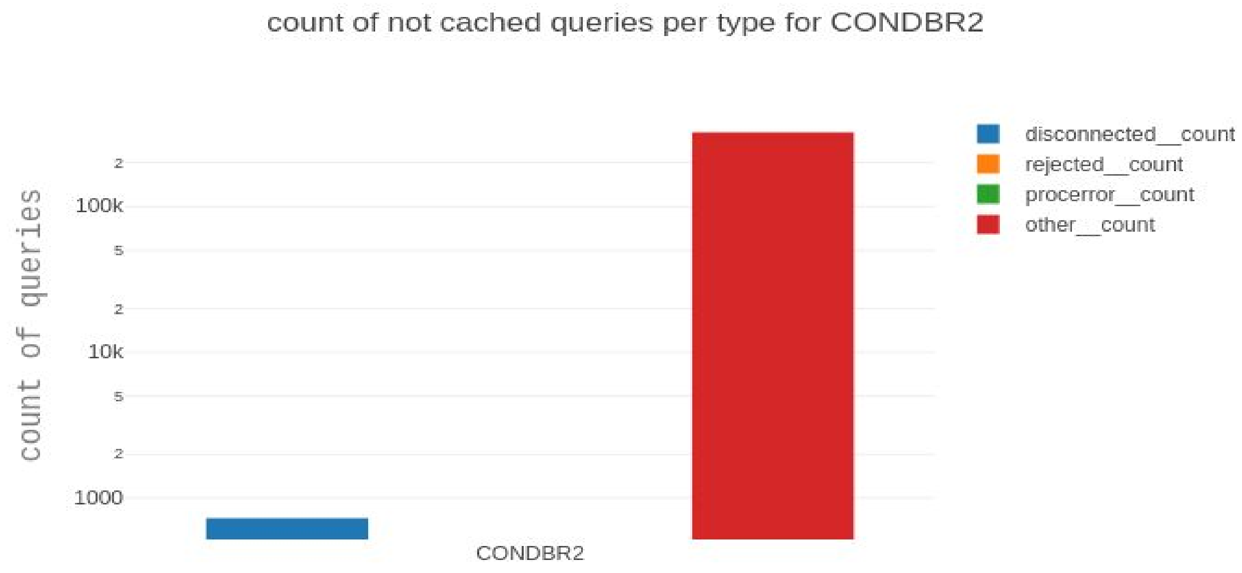


For this overlay task the two COOL instances were requested by the PanDA clients:

- CONDBR2: conditions data for Run 2
- OFLP200: conditions data for MC

The plot shows that the total number of the queries that request OFLP200 is negligible comparing to CONDBR2. That's why we are interested in studying CONDBR2 queries especially the not-cached queries that present almost half of the cached queries.

2. Count of the not cached queries per type (disconnected , rejected, processing error, others):

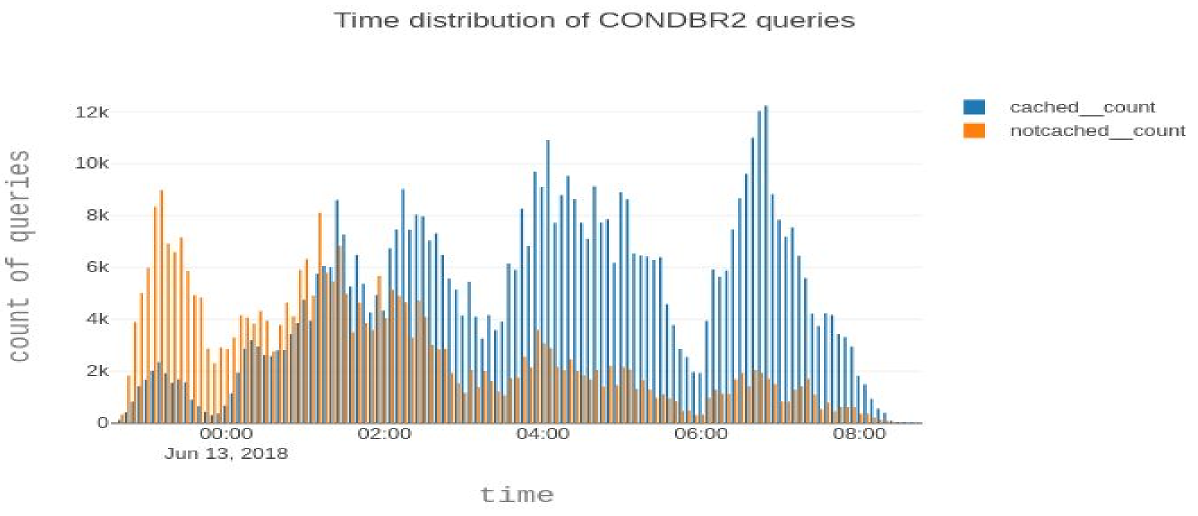


This plot presents the count of the different types of the not-cached queries for the CONDBR2 instance. We notice that there are no rejected or processing error queries for this task. Besides there are 730 disconnected queries which means Oracle database interrupted the connection with the Frontier server, and the majority of the not-cached queries are the queries that ask for the data from the squid-Frontier system for the first time.

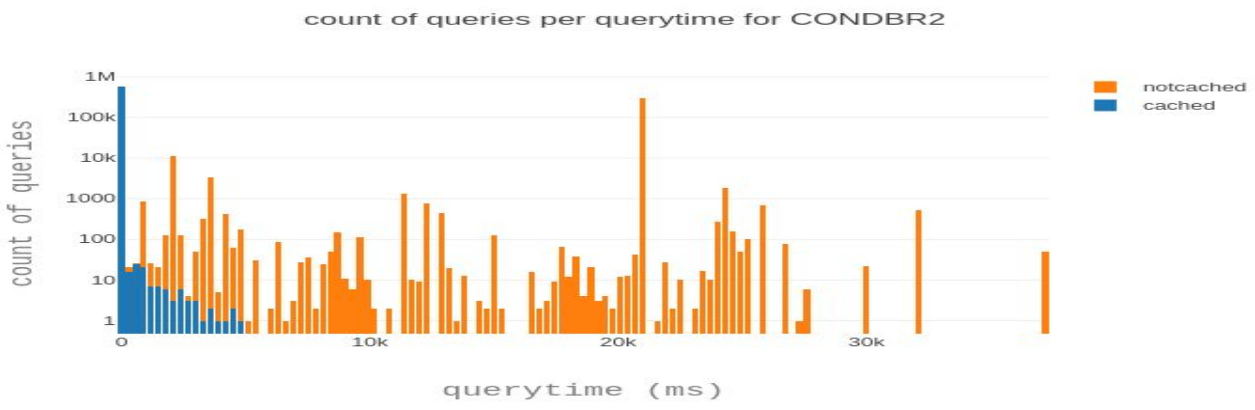
3. Time distribution of the CONDBR2 queries

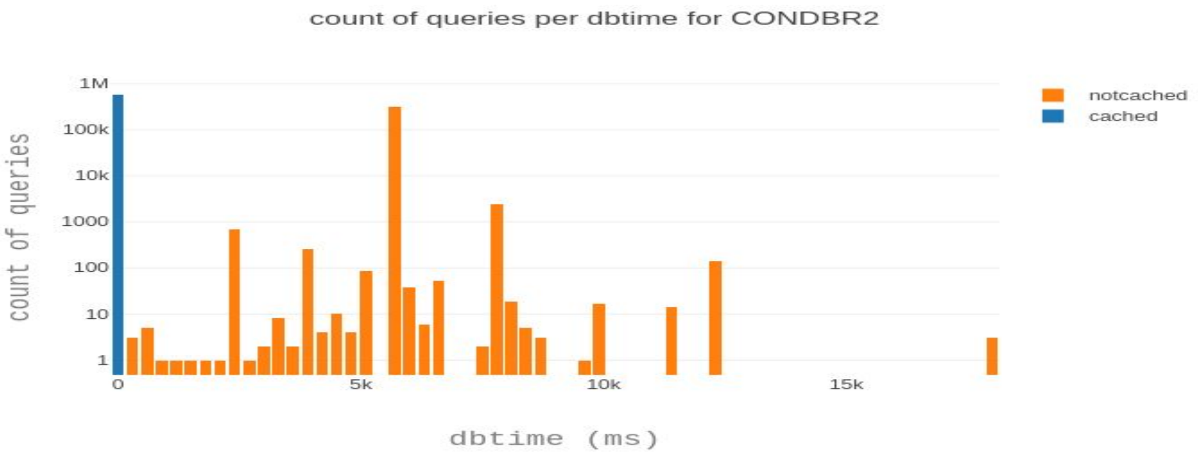
The bar chart illustrates the time distribution of the queries (in time bins of 5 minutes) during the overlay task period that started at June 12th at 22:00 and ended at June 13th at 09:00.

In the first 2 hours, the number of the not-cached queries was increasing until reaching almost 9000 queries at 23:10. So we can say at the beginning almost all the queries were not-cached at the squid-Frontier system. After that, the number of the cached queries was growing up contrary to the not-cached queries. It means that almost all queries were cached at the squids. However, there is an important number of the not-cached queries all along the task was running.



4. Count of the queries per (querytime / dbtime):





These two plots are about counting the queries per:

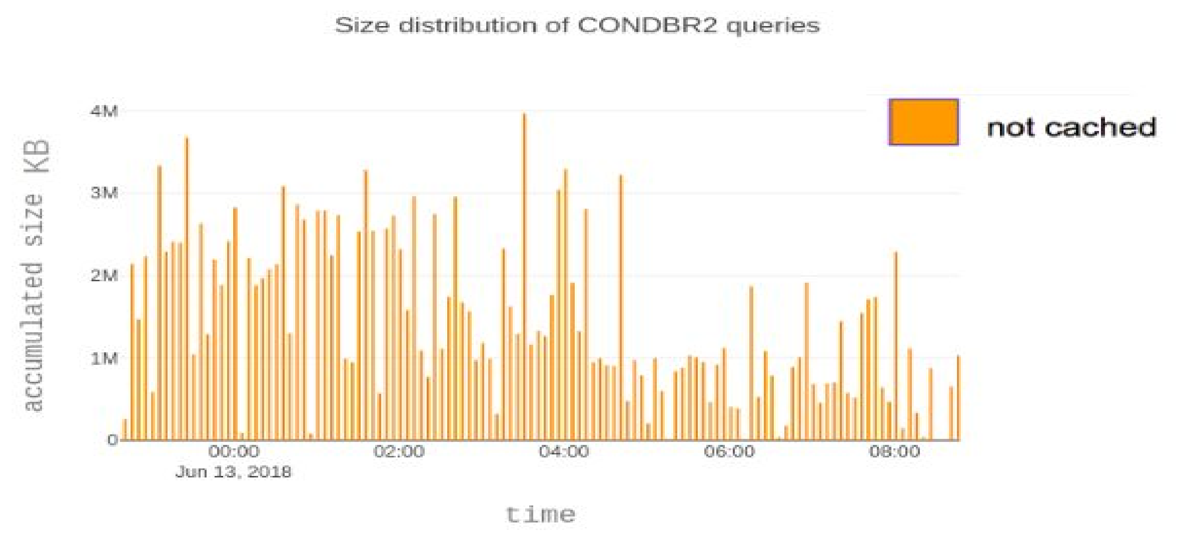
- query time: total time spent since the query starts to be processed until it is completed
- db time: time to get back the data from the Conditions database

The goal is to compare between the db time and the query time and find out the number of the slow queries (queries with query time > 1 ms) as the slow queries could be the cause of the overlay performance.

The first plot shows that the largest number of the cached queries are executed in less than 1 ms. Few cached queries took till 5 ms to terminate contrary to the not-cached queries where the number was going up till reaching 300 K queries with 21 ms as the query time and a small number with 37 ms.

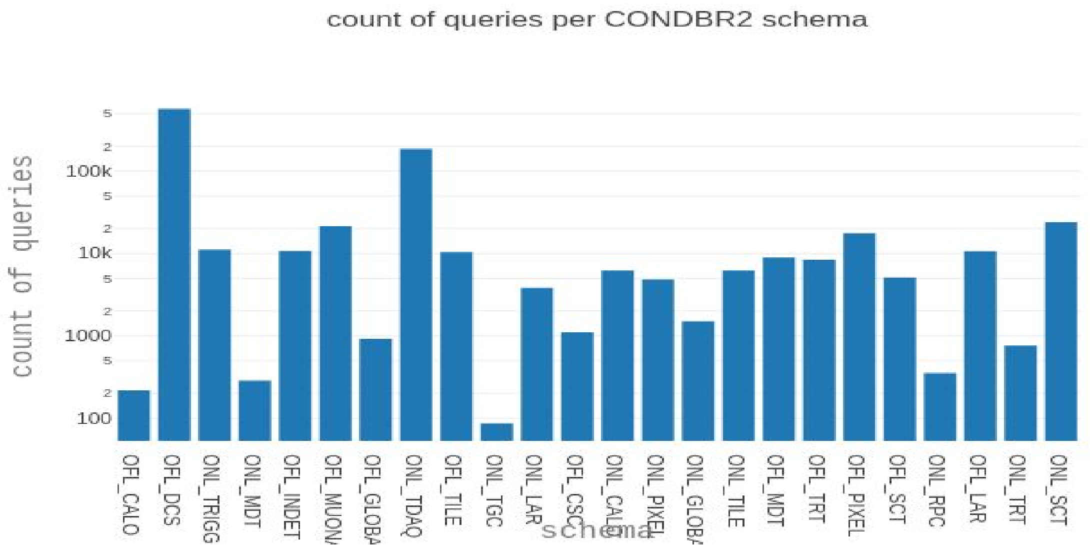
Comparing with the second plot, all the cached queries don't require a db time as they get the data from the squids directly. However, a large number of the not-cached queries exceed 5 ms and some reach 18 ms as the db time.

5. Distribution of the response size of the CONDBR2 queries:



The plot presents the responses size (KB) per time bins of 5 minutes. For 6 hours, the total response size for the queries running every 5 minutes was between 5 MB and 3.68 GB. It reaches the peak at 3:30 where the total size is about 4 GB. So we can see that in the beginning, the system is getting a large amount of data from the COOL database. Once the data is cached at the squids level, the response data size is decreasing. For the cached queries, the response size is 0 KB because the data is retrieved from the squids.

6. **Count of the queries per schema:**



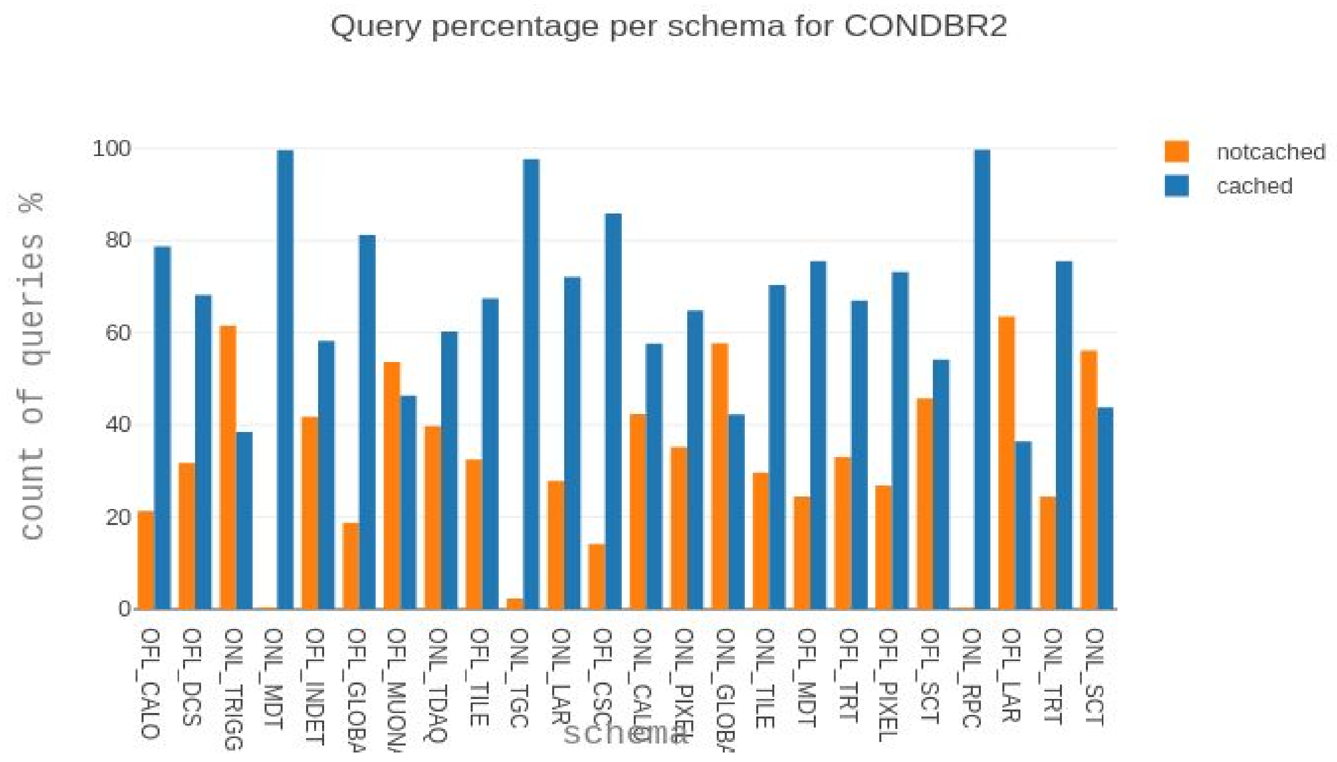
The plot shows the most requested schema in the CONDBR2 instance which is OFL_DCS with more than 572 K queries.

7. **Query percentage per schema:**

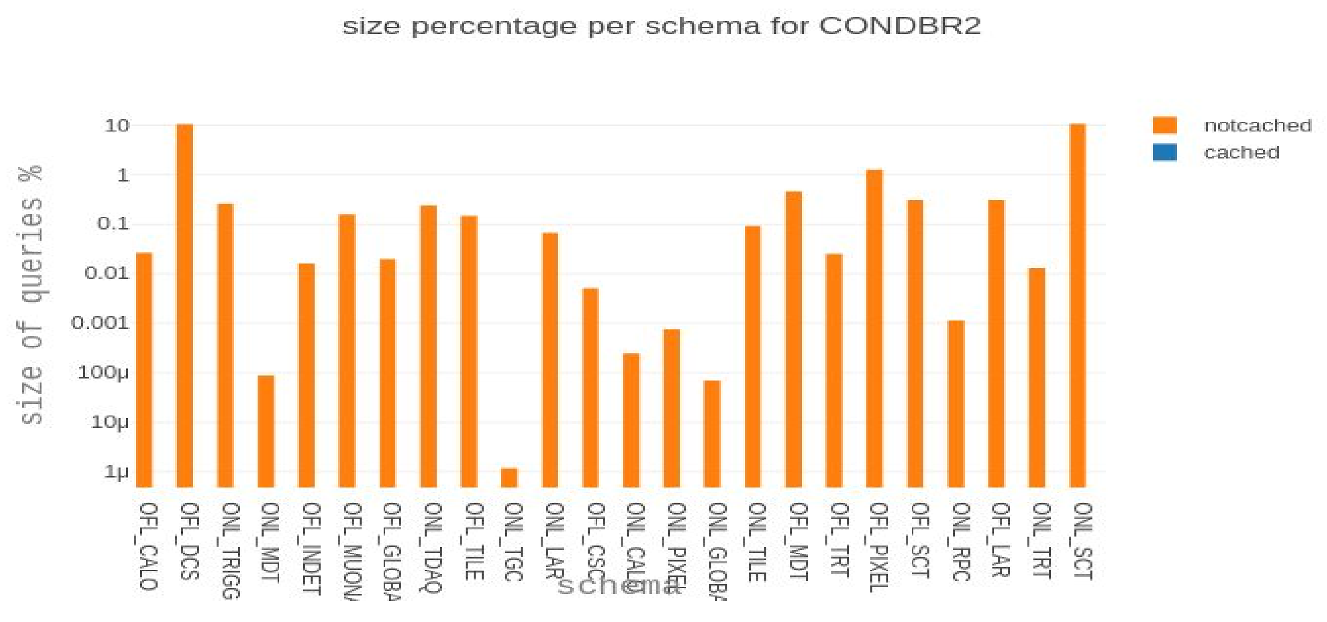
This plot shows for every CONDBR2 schema the percentage of queries (cached, not cached).

To calculate this percentage, we first calculate the number of the total queries for every schema (total_schema). Next, we calculate the number of the cached queries (the number of the not-cached queries) for every schema, then the percentage is (the number of the cached/not-cached / total_schemas) *100.

From this plot we deduce that the LAR schema has more not-cached queries than others (63.59 %) even if it has fewer number of queries than many other schemas such as OFL_DCS. The second largest percentage of the not-cached queries is 61.51 % for ONL_TRIGGER. ONL_RPC and ONL_MDT have the best cached percentage with almost 100%.



8. Size percentage per schema:



In order to better study the database structure and spot possible weaknesses in particular schemas, we are interested in visualizing the response data size distribution per schema. We calculate the percentage of the response data size which is the total response data size for a given schema (not-cached because the response size for the cached queries is 0 KB) divided by the total data size retrieved from CONDBR2.

The plot above is about the percentage of the requested data size from every CONDBR2 schema. As mentioned in the previous plot, OFL_SCT is the most requested schema with 10,61% of total data size, then comes OFL_DCS (10.52%) which is one of the most important schemas in CONDBR2. However, ONL_TDAQ presents only 0.24% of the response data even if it has a larger number of requests.

5) CONCLUSION

Through this project, we have the first version of an information aggregation and analytics system that studies the ATLAS squid-Frontier system behavior. This system provides tools for analyzing and parsing logs data, visualizing data in different plots and comparing between different parameters in order to find out the breaking points of the system. A framework is in place now to study the Conditions data usage by ATLAS jobs and their respective database access queries in detail.

We continue studying the squid caching system. The next step will be to compare the results from the overlay task with a normal reconstruction task. The future step will be setting up a real-time monitoring tool for monitoring the squid-frontier system behavior while the overlay workflow is running together with all the others in the system.

6) BIBLIOGRAPHY AND REFERENCES

[1] D. Barberis et al, “Evolution of grid-wide access to database resident information in ATLAS using Frontier”, J. Phys.: Conf. Ser. 396 052025, 2012.

[2] “Conditions Database” twiki page, https://twiki.cern.ch/twiki/bin/view/AtlasComputing/ConditionsDB

[3] “PanDA” twiki page, https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA

[4] “Conditions Database Metadata” twiki page , https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/ComaPyAmiViews#COMA_Conditions_DB_Metadata

[5] “Frontier Kibana monitoring” twiki page, https://twiki.cern.ch/twiki/bin/view/AtlasComputing/FroNTier#Frontier_Kibana_Monitoring_Analy,

[6] Sloan Ahrens, “Elasticsearch in Apache spark with python”, 2014-10-24, https://qbox.io/blog/elasticsearch-in-apache-spark-python