# Distributed Data Collection for ATLAS EventIndex

Álvaro Fernández Casani
**Carlos García Montoro**
Javier Sánchez

3ª Reunión Presencial VMGRID

3rd June 2019

# Outline

- Latest Achievements.
- Evolution:
  - Back-end.
  - Data Collection.

# Latest Achievements

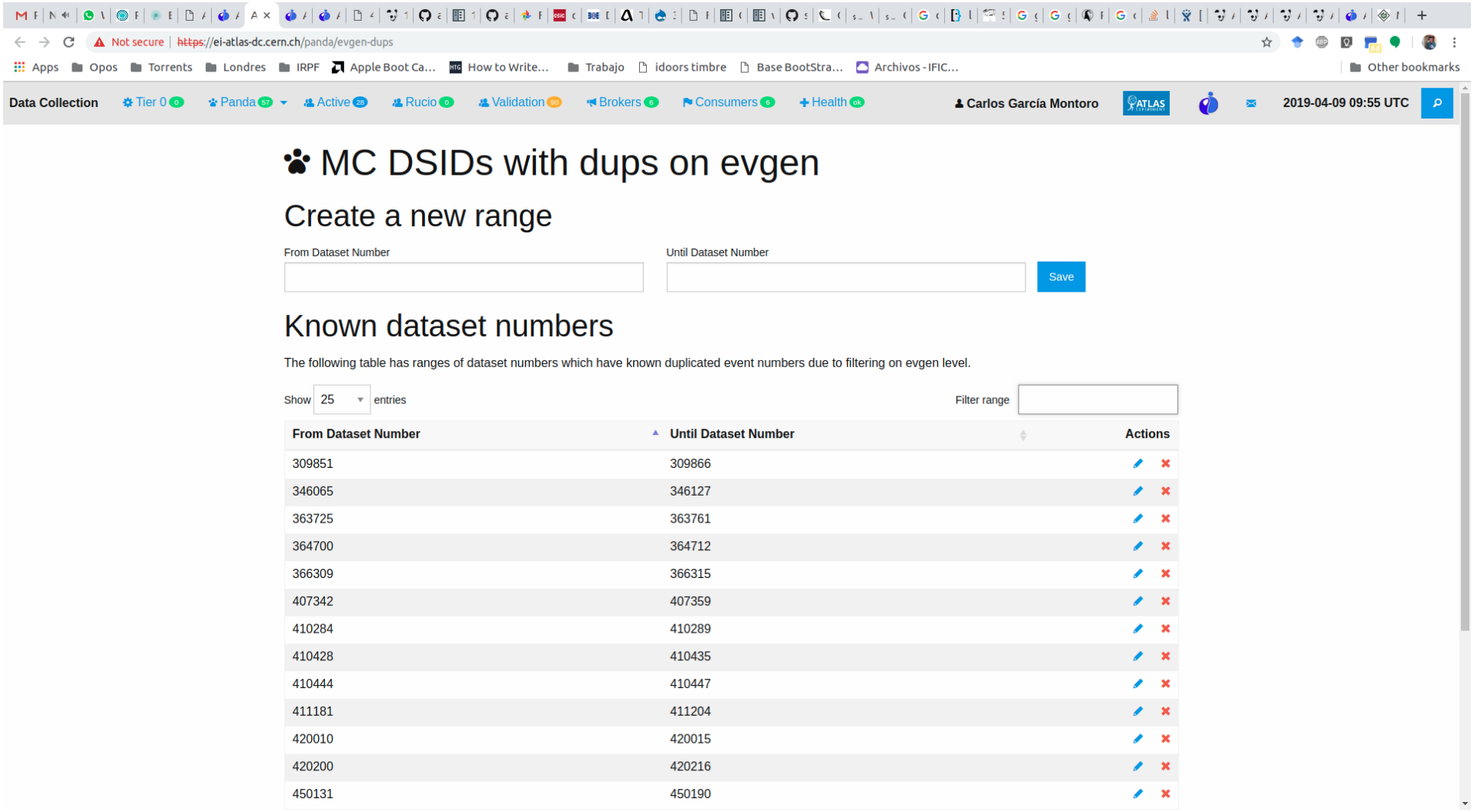
# Data Collection Migration to MySQL

- We have successfully migrated the back-end of the data collection supervisor to MySQL
  - No big issues so far.
  - Big improvement on stability, concurrency and availability.
  - Improvement of flexibility.

# Email Alerts

- The number of emails sent by the supervisor (EIS) due to duplicated event numbers has been drastically reduced.
  - Instead of one email per job with duplicated events, EIS sends just one email per validated dataset for MC.
  - We have limited the number of notified GUIDs with duplicated event numbers to 100.
  - We can provide the full list on demand.
- New alerts have been introduced
  - If Rucio does not provide the number of events of a dataset and any of its files, an email is sent.
- Duplicated event numbers on certain dataset numbers that have an identified problem when filtering on evgen are not notified.
  - We provide a way to maintain a list of dataset number ranges that suffer this problem: https://ei-atlas-dc.cern.ch/panda/evgen-dups

# Duplicated Event Numbers: Blacklist




| From Dataset Number | Until Dataset Number | Actions |
|---|---|---|
| 309851 | 309866 | |
| 346065 | 346127 | |
| 363725 | 363761 | |
| 364700 | 364712 | |
| 366309 | 366315 | |
| 407342 | 407359 | |
| 410284 | 410289 | |
| 410428 | 410435 | |
| 410444 | 410447 | |
| 411181 | 411204 | |
| 420010 | 420015 | |
| 420200 | 420216 | |
| 450131 | 450190 | |

# Duplicated Event Numbers

- We keep helping when information about duplicated event numbers at job level is requested.
  - Data collection supervisor identified that some files in recently requested datasets have only one single event number, which we identify and provide.
  - Dario reported this issue to Doug Benjamin. They are investigating what happened.

# Problems Running EventIndex with new Pilot Code

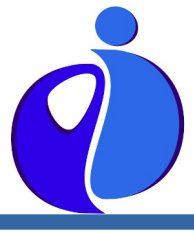
- For some time, data collection monitoring tools were reporting *unknown* jobs.
- Data Collection detected this issue when we were aware of a task that had successfully completed more jobs than those reported to the supervisor.
- After a thorough investigation, we saw in the logs that there were some jobs unable to provide some or all of the following parameters: taskid, jobid and input dataset name.
- We also identified that affected jobs were running a development version of Pilot.

# Problems Running EventIndex with new Pilot Code

- We reported the issue to Panda people
  - https://its.cern.ch/jira/browse/ATLASPANDA-481
- We also reported the issue in our own JIRA:
  - https://its.cern.ch/jira/browse/ATEVTINDEX-79
- We prepared different mechanisms to identify the affected datasets:
  - By means of parsing the supervisor's logs, we could determine some Known tasks with lost messages.
  - But even the supervisor's logs are unaware of tasks that only have affected jobs, therefore we prepared another script to check for unknown datasets that have been attached to a technical container of Event Index.
    - This caused some problems: Is there a way to identify bad datasets which should never be reprocessed? Is it possible / desirable to delete them from the technical container?
- Production tried to declare obsolete the affected tasks and to reprocess affected datasets.

# Trigger Counter

- https://ei-atlas-dc.cern.ch/tc/
- We provide the service for the whole data18.

# Evolution: Back-end

# Hbase and Phoenix

- We are working on how to improve the current back-end of EventIndex, which is based on Hadoop.
- CERN IT advised the use of HBase and Phoneix:
  - CERN IT supports them.
  - Phoenix is built on top of HBase to offer SQL for DDL and DML.

- Following slides present work on the events table.
  - Other tables are also neded to store information of datasets and aggregated / computed / cached information.

# Families

- Families represent related data that is **stored and retrieved together** on the file system. Therefore, all column family members should have the same general access pattern
  - A: Event location (GUID, OID1, OID2).
  - B: Event provenance (in JSON).
  - C: Event description (prescalers l1psk and hltpsk, LB, BCID,...). **Disbanded** to optimize access patterns.
  - D: Level 1 trigger (L1: TBP', TAP', TAV, LB, BCID).
  - E: High Level Trigger (HLT: PH, PT, RS, LB, BCID).

# HBase Row Key Design

- Row Keys identify rows, acting as primary keys.
- Row Keys should have some desirable properties:
  - Small size, since it is written once per row and per family.
  - Identify an event uniquely inside a container for event peeking.
  - Allow searches reading the smallest quantity of consecutive data using ‘range scans’.
  - Avoid sparse insertions into existing row key space.
  - Try to use all regions evenly for reading and writing.
  - When growing, use the Row Key space homogeneously.

# HBase Row Key

- We can have a binary composite key with the smallest number of components:

  **dspid.dstypeid.eventno.seq**

  where:

  - dspid: is an identifier based on:

    < project, runNumber, streamName, prodStep, version >

  - dstypeid is an identifier of the data type
  - seq: allow insertion of duplicate events
- Smallest safe sizes for the RowKey:

  < dspid.dstypeid.eventno.seq > : < 3.2.6.2 > bytes = 13 Bytes per event
- This key allows direct event peeking, range scan over all the events for a given dataset and range scan over all the events for all the derivations streams for the same dspid.
- As a useful side effect, defines a ‘canonical container’ which are all the events that share the same dspid.dstypeid.

# Phoenix Row Key

- RowKey is adapted to Phoenix types and sizes losing some optimizations.
- Dstypeid is subdivided into dstypeid and dssubtypeid

  <(salt).dspid.dstypeid.dssubtypeid.eventno.seq>

  (Byte).Integer.TinyInt.TinyInt.BigInt.SmallInt: <(1).4.1.1.8.2>B
- The slicing function is made internally by Phoenix using:

  salt = hash(rowkey) % bucketNum

  and it is transparent to users creating the necessary subscans if needed
- Phoenix allows the use of RowKey fields in queries but they are stored as one entity in HBase.

# Event Duplicates

- HBase does not distinguish between inserts and upserts.
  - All writes are upserts.
  - Identify duplicates on insertion time would involve trying to read (scan) events first which is very expensive.
- Instead, we use

$$seq = crc16 ( guid:oid1\text{-}oid2 )$$

- Clash probabilities:

| 10 dups | 100 dups | 1000 dups |
|---|---|---|
| 0,06% | 7,25% | 99,95% |

  - Even if we might lose some events, we will detect that there is a duplication problem.

# Tests and Gained Experience

- Phoenix is not as optimal as what can be done with only HBase, but its SQL layer can provide some advantages.
  - Bigger types for Row Key, trigger,…
- We have written some Phoenix User Defined Functions (UDF) to retrieve the self reference of events and their provenance.
- We have also tested different alternatives to store trigger. In the end we will use arrays of native Phoenix types but not repeating triggers for L1.
- Derivation overlaps have been tested using MP with Phoenix.
- TriggerCounter has also been ported to work with the proposed schema and Phoenix.
- Regarding performance some basic tests have been performed.

# Evolution: Data Collection

# Data Collection Evolution

- The evolution of Event Index is closely tied to the evolution of its data collection.
- Is the supervisor prepared for Run 3 and for the evolution of the ATLAS software?
  - Do we have to review our interfaces with other systems?
  - Is the supervisor going to interact with more systems?
- Are we going to extend, modify or replace the back-ends of Event Index?
  - We have already proposed a dataset-centred back-end.

# Dataset-centred data collection

- Data collection should be more dataset-centred. Why?
  - Containers are the logical matter of interest of physicist, but files are grouped into datasets and Panda processes datasets.
  - Since a dataset can be a component of many containers, storing the information per container instead of per dataset might cause a waste of resources, like redundancies and inconsistencies.
  - Most of the information of the containers can be retrieved as the aggregation of their datasets.
- Replace the concept of *Virtual Task* with the concept of *Indexation Request*.

# Indexation Request Ideas

- An Indexation Request (IR) is a request to index a **single name**: either a container or a dataset.
- Rucio is asked about the name N:
  - If N doesn't exist, the IR fails.
  - If N is a container, sub-requests are created. They are linked to the parent request, which life is bound to its children requests. Sub-request are handled mostly just like normal requests.
  - If N is a dataset:
    - If N has never been indexed, process it.
    - If N is being indexed, or it is already indexed, be clever: Check whether N has differences with respect to its last indexation.
      - If there are no differences, the same task can be considered the valid one for attending this request.
      - Otherwise, reprocess the whole dataset or compute the differences.

# Finer Indexation

- When a dataset has changed, probably there is no need to fully index it again.
  - **Deletions don't require re-indexation**, just the deletion of the deleted GUIDs and events in the back-end.
  - **Additions** of files only require the **reprocess** of the **added files**.
  - We can create custom datasets with only the added or modified files and process it instead of the original one.
- Of course, any cached or computed piece of information has to be reprocessed taking into account the added and deleted files.

# Looking for changes in datasets

- *updated_at* is part of the dataset's metadata.
- *list_content_history* is a method that shows the deletions of datasets in containers.
  - It doesn't provide information about when the content was deleted.
  - Does it work with datasets and files?
  - No information about additions or replacements.
- Has Rucio a better way to identify the history of a dataset?
  - Versions of the dataset?
  - Changes between versions?
- If not, we can determine the changes:
  - By storing the version used to validate and consume the dataset.
    - Partially reindexed datasets are difficult to manage in this approach.
  - By looking for the GUIDs known by the supervisor through messages, and comparing them with those in the current version of Rucio: Processing those that are only in Rucio and deleting in the back-end those that are only in the supervisor.
    - Idea: Adler32 to determine whether the GUID has changed.

# Indexation Request States

- Received: Received by the supervisor. Rucio will be asked about it.
- Failed: Not fulfilled.
- Task Requested: A task to process a dataset has been requested.
- Task Running / Panda: Being processed.
- To validate: Check whether all files in the current version have been processed at least once.
- Valid: All files have been processed. Create consumer objects.
- Consuming.
- Done: IR successfully processed.
- Partial: IR available in some requested back-ends, but not in all of them.
- Superseded?: Complete, but the dataset has changed since it was indexed in this request. Its task still has valuable indexing information.
- Obsolete?: There is other full indexation task of the dataset that replaces this one. The results of the task shouldn't be used.
  - Do we need or want superseded and obsolete states?
  - Are superseded and obsolete states of the request or of the task?

# Concurrency and Indexation Requests

- Actually, supervisor doesn't need to know the content of a dataset until it has to validate it.
- Idea: Retrieve metadata (again) and content (maybe for the first time) when the dataset has to be validated.
- Dataset might have change at that point but:
  - Deletions with respect to what has just been processed can be ignored. They should only update the back-end if the deleted info was there.
  - We can process just the files that were added and **validate the dataset** considering all the tasks.
- Therefore, if there are new requests for the same dataset while it is not validated, the new requests can be served with the ongoing ones.
- If the dataset is valid and being consumed, then new requests may wait for full consumption for a back-end before new consumption objects  for that back-end are created.

# Pluggable Consumers

- How many back-ends will have Event Index?
  - Hdfs
  - Hbase with or without Phoenix?
  - Oracle
  - Test environments and or back-ends?
- Making the consumption of indexed datasets more pluggable allows to have many  back-ends with a single indexation and validation process.

# Pluggable Consumers

- Supervisor should know consumer types and associated brokers.
- Validation just validates and prepares *consumption objects*, but it no longer tracks consumption.
- Datasets can be in different states for different back-ends.
- Once **a dataset** is validated, a *consumption object* should be created for each back-end in which is going to be imported.
- Consumption objects are queued and have states.
- To achieve load balance in consumers, the supervisor can handle a number for each pair of consumer type and feasible broker, like pending events to be consumed, to determine which broker should be used.

# Consumption Object States

**Pending**: Not yet sent to a broker consumer.

**Sent**: Sent to a broker. Waiting for a consumer in the broker.

**Consuming**: Being consumed by a consumer. The consumer_id should be marked when consumption begins.

**Failed**: The consumption failed and it should be considered an error from the point of view of the dataset/task. It might be retried automatically or manually. It might be declared obsolete. It might require a new validation and its related consumption object only for this consumer type.

**Obsolete?**: The consumption failed but it shouldn't be considered a current and actual error from the point of view of the dataset. There might be another consumption object that replaces this one, or it just shouldn't be retried / alerted.

**Consumed**: Successfully consumed.

*Questions:*

- Do we really have to store consumed objects?
- Can a consumed consumption object be superseded?
- *Superseded* is a state or *obsolete* is also useful for this situation.

# Other Questions for / related to Panda

- Settle with Panda how the supervisor should retrieve the information of ongoing tasks:
  - HTTP + JSON should be a must to decouple both systems.
  - Currently we perform many requests of a single task each cycle. Do they prefer a single request of many tasks per cycle?
  - Do we need the detailed information that is retrieved now?
    - If yes, what about many light requests and a single full request when the task reaches a final state?
    - If no, what do we really need:
      - Task name.
      - Dataset name.
      - Task status.
      - Jobs? Useful to detect the problem with unknow tasks, jobs and dataset.
      - Timestamps?
  - Can panda provide the Rucio *updated_at* of the dataset that it processes?

# Other Questions for / related to Panda

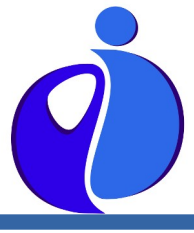
- What is the best strategy to run tasks in Panda?
  - Right now, we attach datasets to technical containers.
  - Is it possible to directly create the task with its parameters?
  - What are the advantages and disadvantages of both strategies?

# Other Questions for / related to Tier 0

- Confirm that they are happy with how the supervisor retrieves the information of tasks.
- Confirm that they are not going to change their endpoints.
- Ask them if they can add the year of the task in the information that they provide.

# Other Questions for / related to Rucio

- Is updated_at reliable enough to distinguish two versions of the same dataset?
  - It must change when files have been added, deleted or modified.
- Confirm that they won't change their python client.
- What are the capabilities of Rucio to store the history of a dataset?
  - Deletions seem to go to content-history, but no info about when it occur is provided.
  - Is there a call to ask about additions and modifications?
  - Can Rucio provide the differences w.r.t. files between two versions or should the supervisor be full aware of the versions that are consumed?



# Thank you!