HBase / Phoenix-based Data Collection and Storage for the ATLAS EventIndex

Carlos García Montoro[1*], **Javier Sanchez**[1], **Dario Barberis**[2], **Santiago González de la Hoz**[1], **José Salt**[1]

[1]Instituto de Física Corpuscular (**IFIC**) CSIC – University of Valencia, Valencia, Spain
[2]University of Genoa; **INFN** Genoa, Italy
[*]Corresponding Author

The ATLAS EventIndex

Is the catalogue of all ATLAS real and simulated events.
- ***Deployment and Operation of the ATLAS EventIndex for LHC Run 3*** describes its architecture.
- This poster is about its **data collection**, its **data storage**, and their evolution for Run 3.

Evolution

1/ **Original Implementation** at the beginning of Run 2:
- Back-end based on HDFS files organized per dataset.
- Data Collection exclusively based on messaging from producers to consumers.
  - Cumbersome.
  - Production is not re-playable.

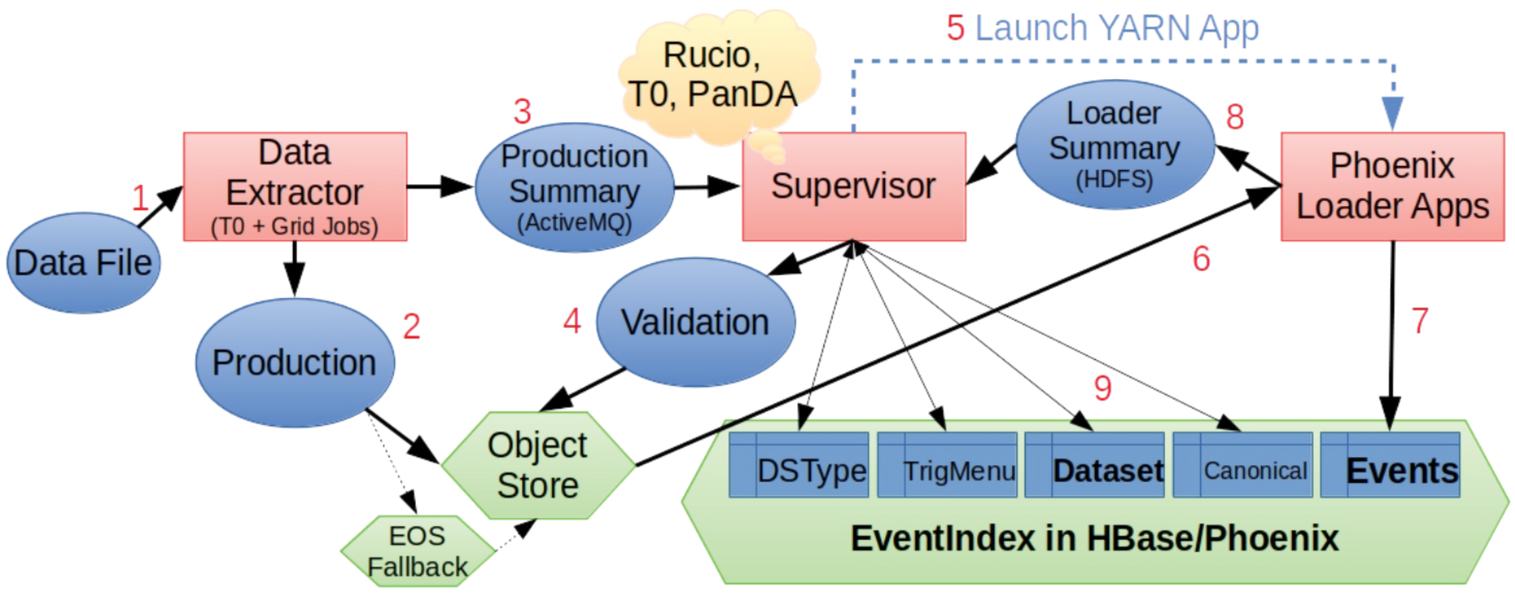
2/ **ObjectStore Implementation** for Run 2:
- The Data Collection **Supervisor** is introduced to orchestrate the whole data collection.
  - Accountability of data collection and validation.
- Amazon-S3-like **ObjectStore** at CERN replaces pure messaging architecture.
  - Simple and scalable.
  - Production can be stored, backed up, and replayed.

3/ **HBase/Phoenix Implementation for Run 3**:
- Back-end based on **HBase**, compatible with **Phoenix**.
  - Improved data structures.
  - Simplified management.
  - Has a SQL interface.
- **Spark/Scala Loaders** replace consumers.
  - Better performance.
  - Improved scalability and adaptability.
  - Open Source Industry standards.

The Data Collection

Is the process that:
1) Extracts the metadata to index.
2) Validates it, assuring correctness and completeness.
3) Loads the metadata into the EventIndex back-end.

Data Flow of Data Collection

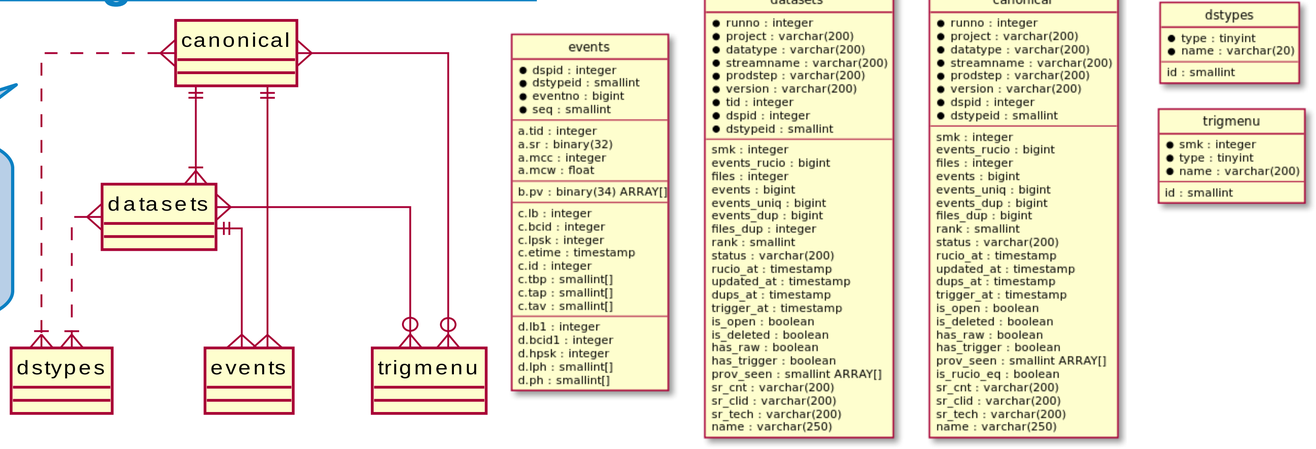


EventIndex Records

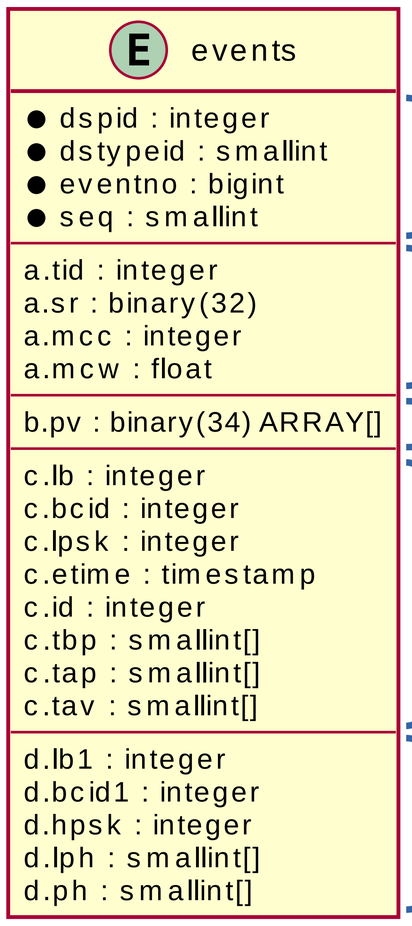
- Event Records: **~533 Billion**
- Files indexed: **~32 Million**
- Datasets indexed: **~279,000**

Storage Logical Architecture

NoSQL:
Relations are inferred but not enforced.



Event Record Table



Very optimized primary key
- Balanced use of all regions and region servers.
- Locality of events of each dataset:
  dspid and dstypeid identify datasets.
- Locality of derivations for overlaps:
  Same dspid, different dstypeid.
- Seq is a CRC16 to record duplicates, if any.

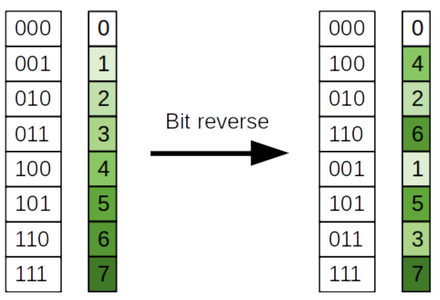
Compact
- Primary key is small, 128 bits, and identifies the dataset with a pair of numbers, no names.
- Trigger stored as smallint arrays, no names.

Families to read just what is needed on each use case:
a) For event picking.
b) For provenance.
c) For L1 trigger operations (count, overlap,…).
d) For L2 and HLT trigger operations.

Compatible with Phoenix but without using its exclusive features.
- Depend only on HBase, but not on Phoenix.

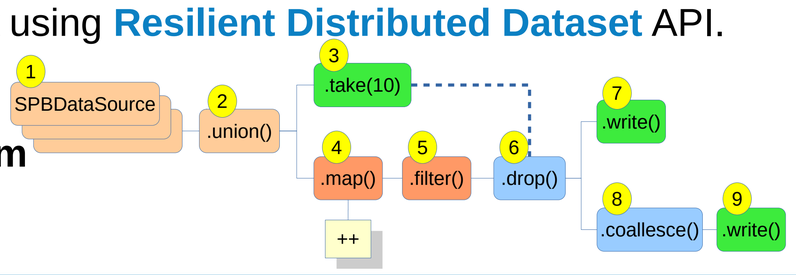
Event Record Primary Key

- Dataset names are too long.
  - Use artificial dataset **identifiers** instead.
  - **dspid** and **dstypeid** are generated by the supervisor by means of the autoincremental feature of its RDBMS.
- Monotonically increasing keys are undesirable in HBase.
  - Reverse the bits of dspid to populate all the key space uniformly.
  - Reserve the first bit to distinguish between data and mc.
  - Reserve bits 2 to 4 for versioning.



Phoenix Loaders and Importers

Write the EventIndex data into HBase/Phoenix event table.
- Scala with Spark using **Resilient Distributed Dataset** API.
  - **Lazy**
  - **In phases**
  - **High parallelism**
  - **Resilient**



GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN

MICINN supported this work under grant **PID2019-104301RB-C21**