The ATLAS EventIndex using the HBase/Phoenix storage solution

Evgeny Alexandrov[1], Igor Alexandrov[1], Dario Barberis[2], Luka Canali[3], Álvaro Fernández Casaní[4], Elizaveta Cherepanova[1*], Elizabeth Gallas[5], Carlos García Montoro[4], Santiago González de la Hoz[4], Julius Hrivnac[6], Andrei Kazymov[1], Mikhail Mineev[1], Fedor Prokoshin[1], Grigori Rybkin[6], Javier Sánchez[4], José Salt Cairols[4], Miguel Villaplana Perez[4], Alexander Yakovlev[1]

1: JINR Dubna, 2: Univ./INFN Genova, 3: CERN, 4: IFIC Valencia, 5: Univ. Oxford, 6: IJCLab Université Paris-Saclay

GRID'2021

July 5-9, 2021

* elizaveta.cherepanova@cern.ch

Outline

- Introduction
- EventIndex records and use cases
- EventIndex architecture
  - Data production
  - Data collection
  - Data storage
- EventIndex evolution (HBase+Phoenix)
- Data structures of the new system
- New system performance
- Conclusion

Introduction

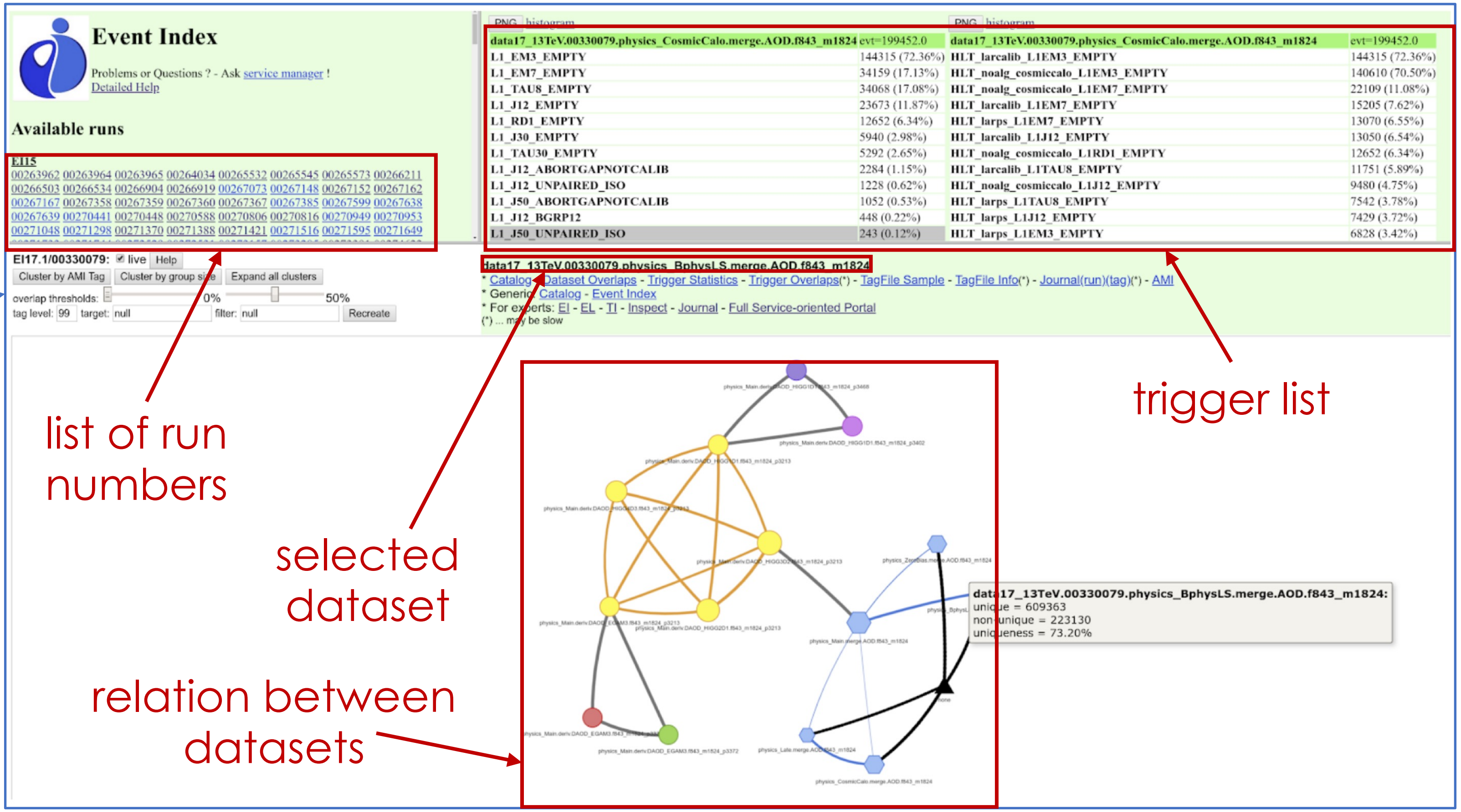
The ATLAS experiment produces **several billion events** per year (real and simulated data)

The ATLAS EventIndex (EI) – global catalogue of all events collected, processed or generated by the ATLAS experiment at the CERN LHC accelerator

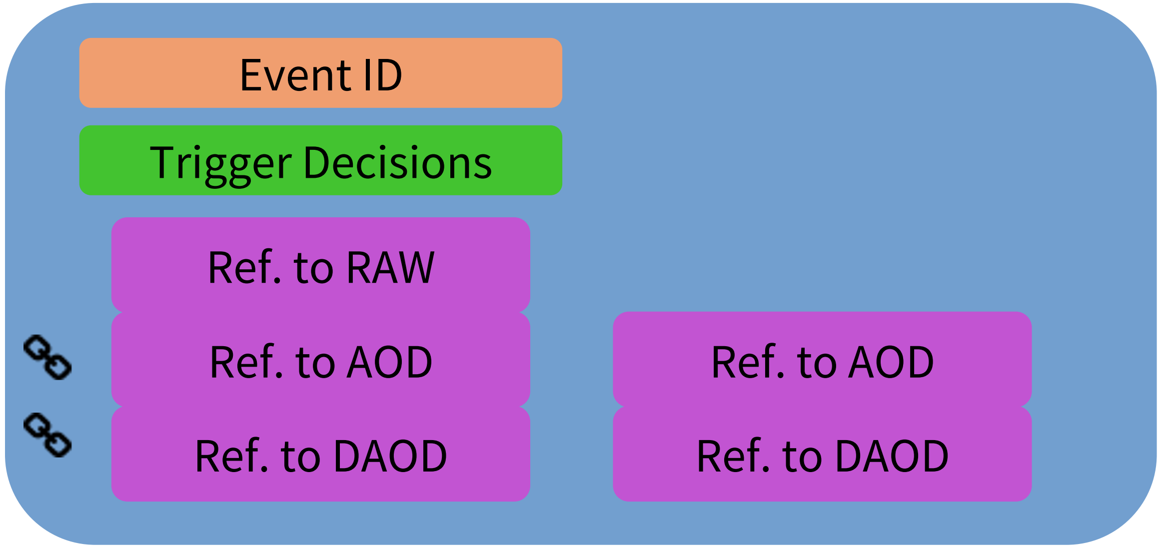
The system provides:

- a way to collect and store event information using modern technologies
- various tools to access this information through command line, GUI and RESTful API interfaces
- an indexing system that points to these events in millions of files scattered through a worldwide distributed computing system

Web interface: https://atlas-event-index.cern.ch/EIHadoop/



EventIndex records

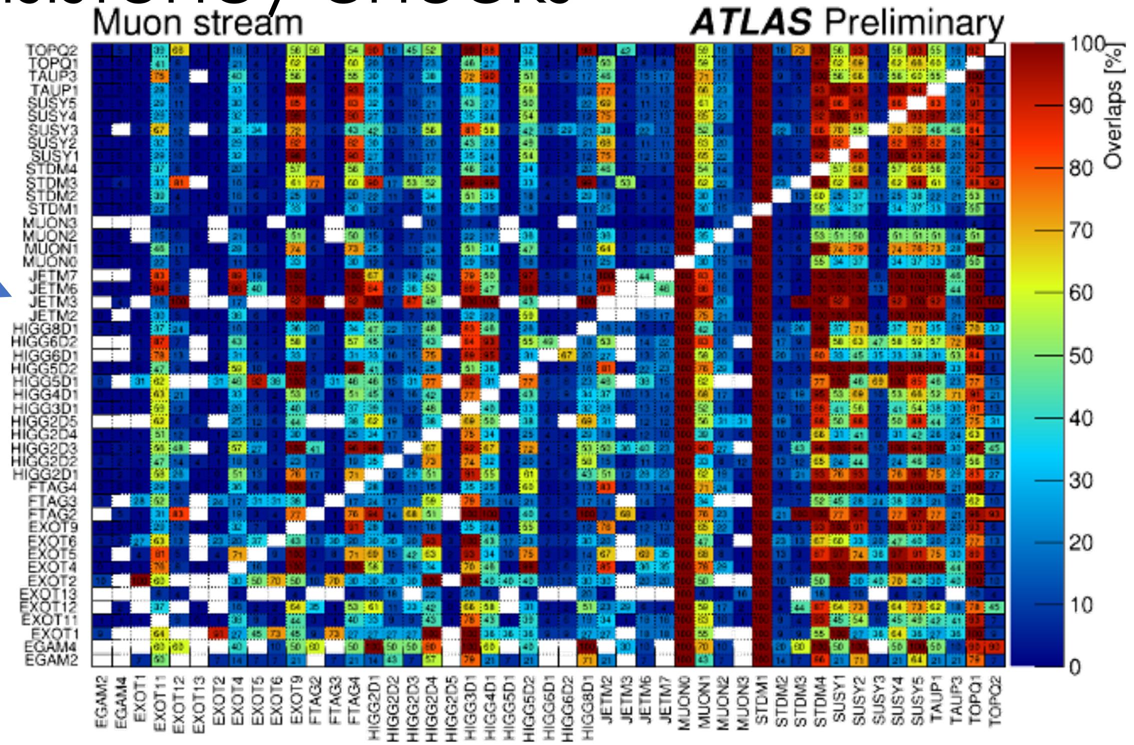


- Event identifiers
  - Run and event number
  - Trigger stream
  - Time stamp
  - Luminosity block
  - Bunch Crossing ID (BCID)
- Trigger decisions
  - Trigger masks for each trigger level
  - Decoded trigger chains (trigger condition passed)
- Location information
  - GUID (Globally Unique Identifier)

More information: https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/EventIndexUseCases

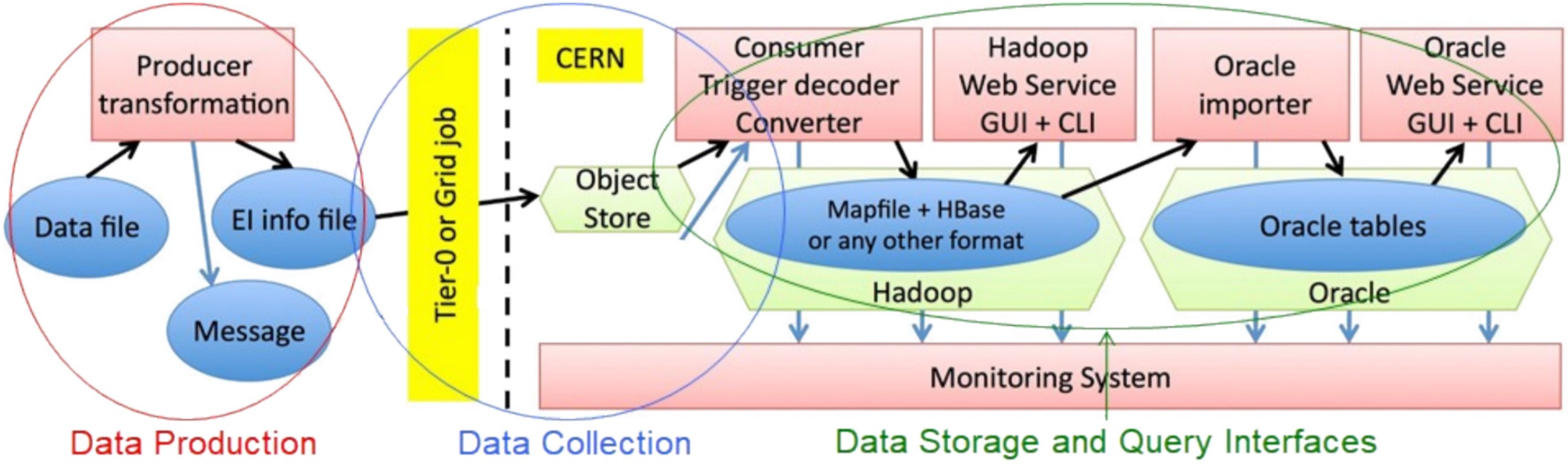
Use cases

- Event picking
  - get an event in the specific format and processing version
- Count or select events based on trigger decisions
- Production completeness and consistency checks
  - data corruption, missing and/or duplicated events
- Trigger chain overlap counting
- Derivation overlap counting
- Dataset Browsing
  - find a dataset of interest
  - get the info about this dataset
  - dataset inspection



More information: https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/EventIndexUseCases

EventIndex Architecture



Data production:

- Extract event metadata from files produced at Tier-0 or on the Grid

Data collection:

- Transfer EI information from jobs to the central servers at CERN

Data Storage and Query Interfaces:

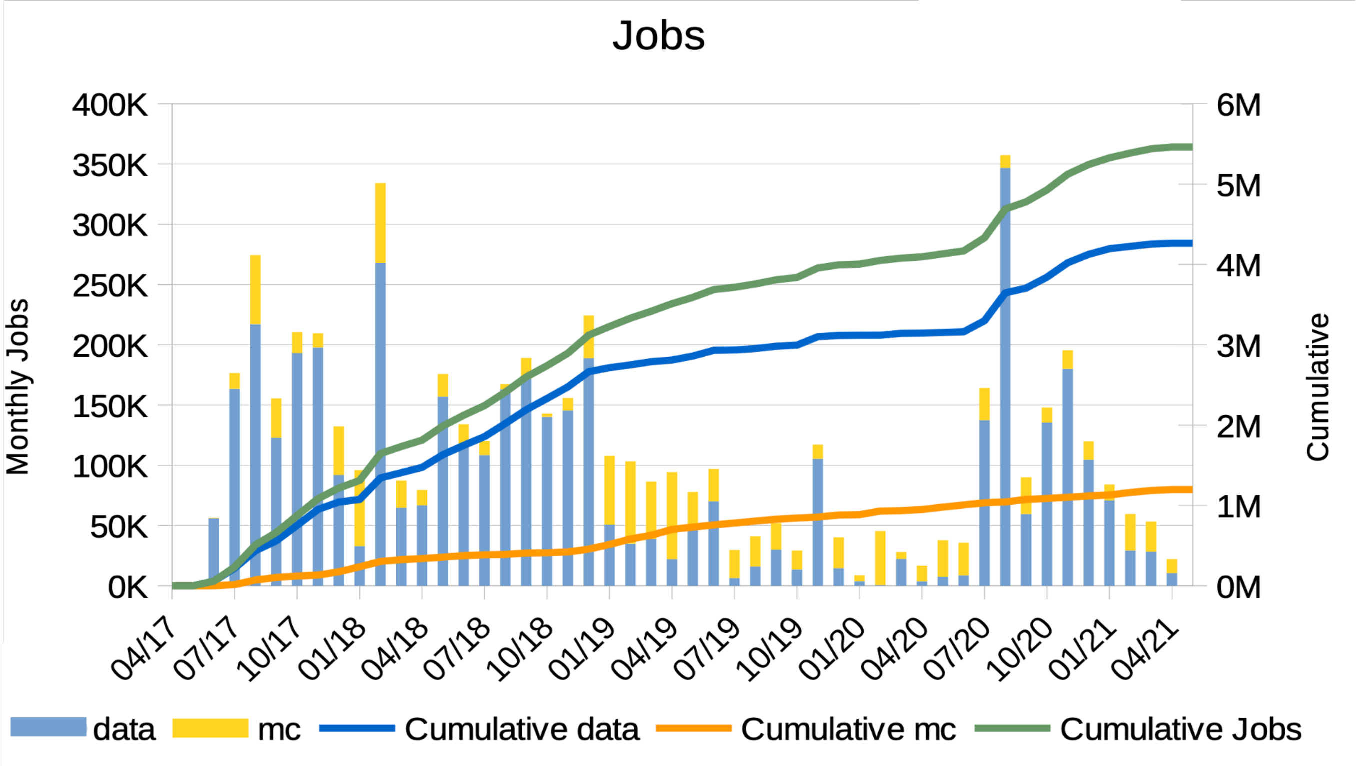
- Provide permanent storage for EventIndex data
  - full info in Hadoop (one dataset = one 'MapFile')
  - reduced info (real data only, no trigger) in Oracle
- Fast access for the most common queries, reasonable time response for complex queries

Monitoring:

- Track of the health of servers and the data flow

Data production

- Tier-0 jobs index merged physics AODs (Analysis Object Data), real data DAODs (derived AOD) and EVNTs (simulated events) collecting also references to RAW data
- Grid jobs collect info from datasets as soon as they are produced and marked "complete" in ATLAS Metadata Interface (AMI)
- Other data formats (HITS, simulated DAOD etc.) can be indexed on demand
- Continuous operation since spring 2015



Data collection

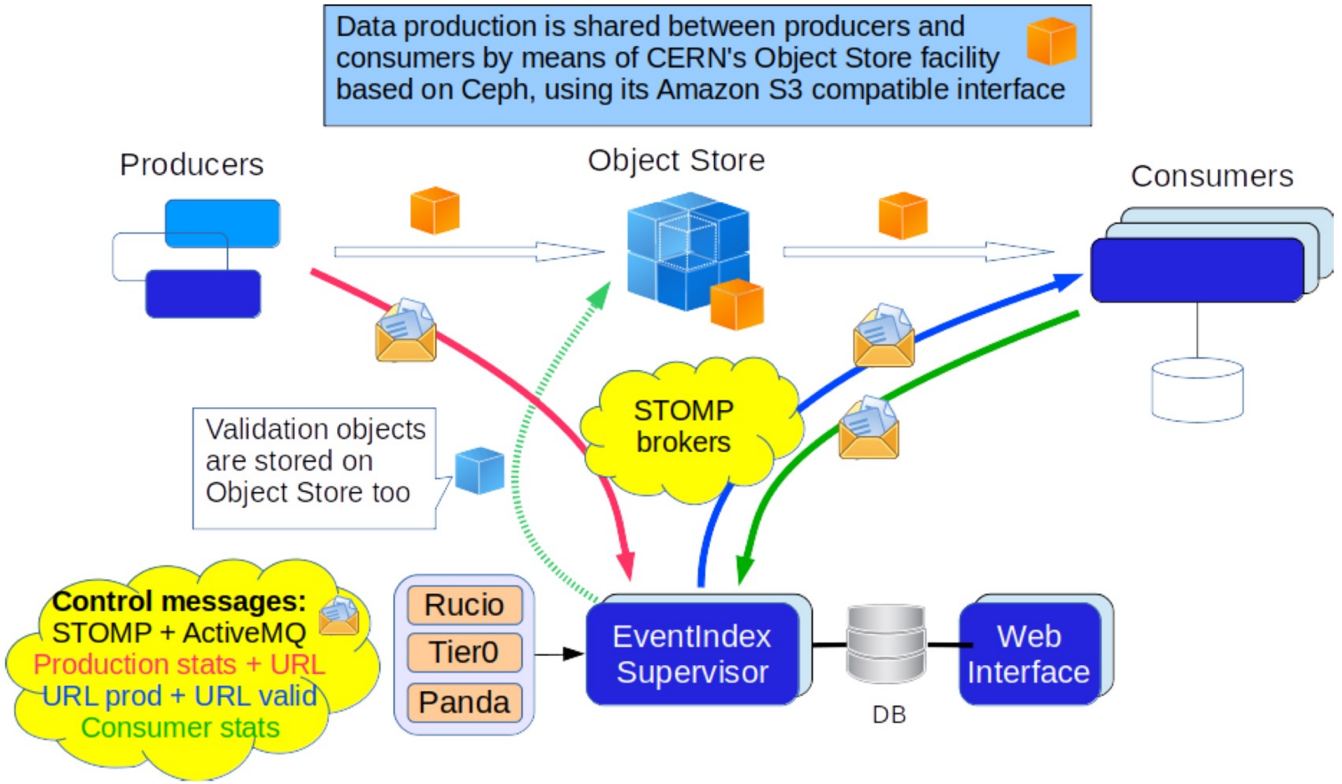
Producer: Athena Python transformation, running at Tier-0 and grid-sites. Indexes dataset

- data and produces an EventIndex file
- EI Information is se t by each job as a file to the ObjectStore at CERN (CEPH/S3 interface) as intermediary storage
- **Google protobuf** data encoding (compressed)

Supervisor: Controls all the process, receives processing information and validates data by dataset

- Signals valid unique data for ingestion to Consumers.
- Operated with a web interface

Consumers: Retrieve ObjectStore data, group by dataset and ingest it into Hadoop storage

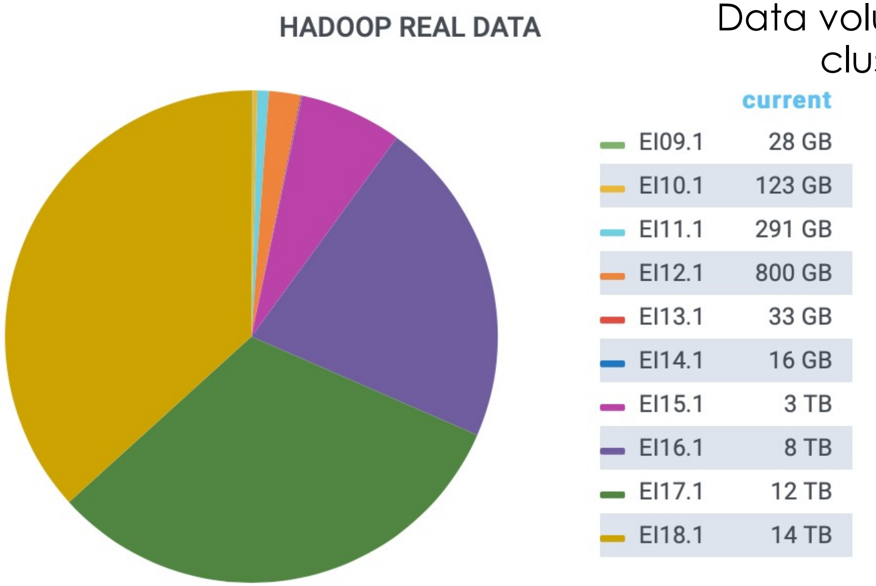


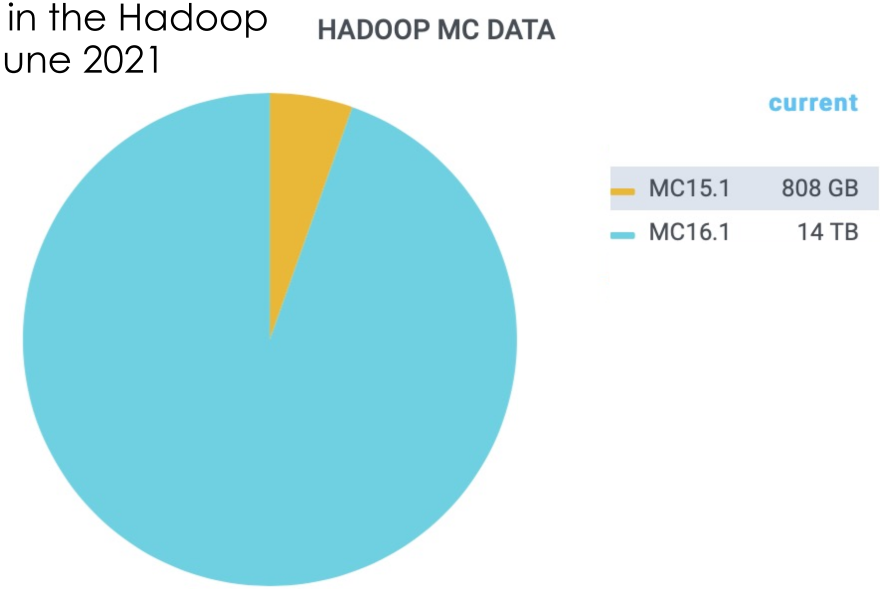
Data storage - Hadoop

→ baseline storage technology

- Every dataset is stored as "MapFile" at HDFS (Hadoop distributed file system)
- easy to search and retrieve data
- has a relational database Hbase on top
  - internal dataset catalogue for the Hadoop Mapfiles
  - server for the Event Lookup service
- CLI, RESTful API and GUI interfaces available
- free and open source

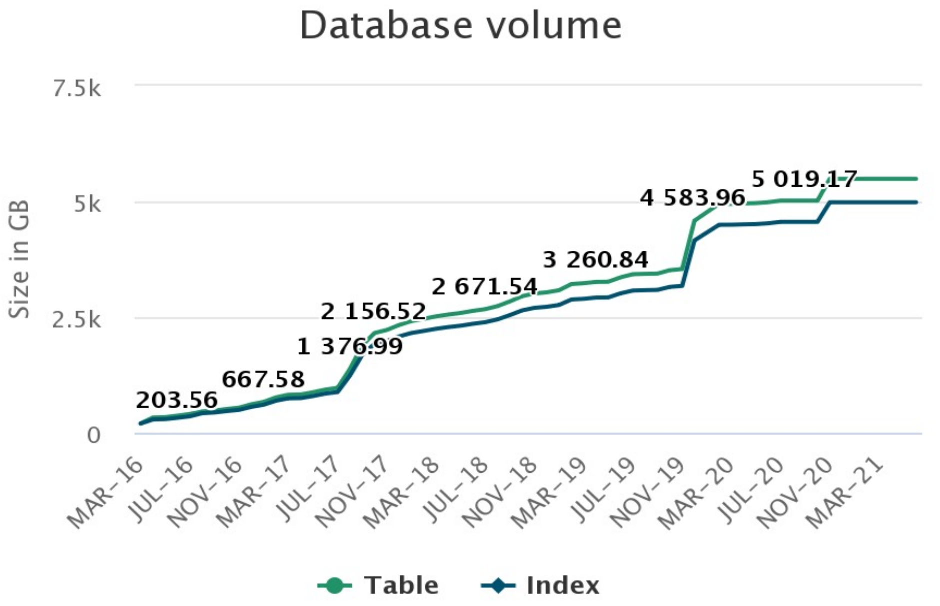
Data volume used in the Hadoop cluster, mid-June 2021





Data storage - Oracle

- Simple storage schema with dataset and event tables
- Exploiting the relational features of Oracle
- Filled with all real data, only event identification and pointers to event locations
  - Optimized for event picking
  - Very good performance also for event counting by attributes (LumiBlock and bunchID)
- Connection to the ATLAS RunQuery and AMI databases to check dataset processing completeness and detect duplicates
- Easy calculation of dataset overlaps
- GUI derived from COMA database browser to search and retrieve info



Disk storage size used by EventIndex tables in the Oracle cluster

EventIndex evolution

Existing storage implementation shows the best that was available at time the project started (2012-13), but:

- Lots of duplicated Mapfiles on HDFS (Hadoop distributed file system): the same event across each processing step (RAW, ESD, AOD, DAOD, NTUP) is physically stored at different HADOOP HDFS files

Future storage implementation:

- One and only one logical record per event: Event Identification, Inmutable information (trigger, lumiblock, ...)
- For each processing step:
  - Link to algorithm (processing task configuration)
  - Pointer(s) to output(s)
  - Flags for offline selections (derivations)

EventIndex evolution

Existing EI version:

- Can operate with Run2 rates

  ALL ATLAS processes:
  - ~30 billion events records/year
  - up to 350 Hz on average

  (This is update rate through the whole system :all years, real and simulated data)
- Read 8M files/day and produce 3M files

Future EI generation:

- Needs to be scaled due to the upgrade to HL-LHC and expected trigger rates
  - At least half an order of magnitude for Run3 (2021-2023):
    35 B new real events/year and 100 B new MC event/year
  - An order of magnitude for Run4 (2026-2029):
    100 B new real events and 300 B new MC events per year

EventIndex evolution

HBase+Phoenix

HBase:

- the Hadoop database, a distributed, scalable, big data store
- organizes data into tables
- belongs to NoSQL database family
- various possibilities for SQL on Hbase
- Has RowKey design:
- Have a small size
- An event uniquely identified
- Allow searches reading the smallest quantity of consecutive data using ‘range searches’
- When growing, use the RowKey space homogeneously

Apache Phoenix:

- takes and compiles SQL queries into a series of HBase scans
- direct use of the HBase API, along with coprocessors and custom filters
- produces regular Java DataBase Connectivity result sets
- HBase RowKey design is well adapted to Phoenix’s types and sizes

Evolution of the Data Collection system

required due the usage of the HBase/Phoenix storage

Producer: upgrade for better performance working with the latest software

Supervisor upgrade:

- Ability to control the creation of indexing tasks on PanDA
- Increased granularity in the indexation process: can re-index only failed files of a dataset without re-indexing others
- Increased granularity of the data consumption: can perform validation of the dataset during its production

Consumers: will write to HBase/Phoenix using Spark jobs

Data structures in HBase/Phoenix

- HBase tables works best for random access – event picking
- A range of data can be accessed by scanning data - analytic use cases

Row Keys:

- Should include the most needed info
- Have minimal possible size

⇒Chosen structure:

dspid.dstypeid.eventno.seq

- dspid: an identifier for the dataset name
- dstypeid: dataType
- eventno: event number
- seq: unique value in case of dataset name and EventNumber duplication

```
CREATE TABLE IF NOT EXISTS events
(
    dspid          integer              NOT NULL ,
    dstypeid       smallint             NOT NULL ,
    eventno        bigint               NOT NULL ,
    seq            smallint             NOT NULL ,

    a.tid          integer                       ,
    a.sr           binary(24)                    ,
    a.mcc          integer,
    a.mcw          float,

    b.pv           binary(26) array              ,

    c.lb           integer                       ,
    c.bcid         integer                       ,
    c.lpsk         integer                       ,
    c.etime        timestamp                     ,
    c.id           bigint                        ,
    c.tbp          smallint array                ,
    c.tap          smallint array                ,
    c.tav          smallint array                ,

    d.lb1          integer,
    d.bcid1        integer,
    d.hpsk         integer,
    d.lph          smallint array,
    d.lpt          smallint array,
    d.lrs          smallint array,
    d.ph           smallint array,
    d.pt           smallint array,
    d.rs           smallint array,

    CONSTRAINT events_pk PRIMARY KEY
    (dspid, dstypeid, eventno, seq)
) DATA_BLOCK_ENCODING='FAST_DIFF', COMPRESSION='SNAPPY';
```

Data structures in HBase/Phoenix

- HBase tables works best for random access – event picking
- A range of data can be accessed by scanning data - analytic use cases

Data families:

- A: Event location (and MC info)
- B: Event provenance
- C: Level 1 trigger (L1)
- D: High Level Trigger (EF of HLT) and L2 for Run1 data

Data import can be performed with Map/Reduce or Spark jobs

```
CREATE TABLE IF NOT EXISTS events
(
    dspid          integer           NOT NULL ,
    dstypeid       smallint          NOT NULL ,
    eventno        bigint            NOT NULL ,
    seq            smallint          NOT NULL ,

    a.tid          integer                    ,
    a.sr           binary(24)                 ,
    a.mcc          integer,
    a.mcw          float,

    b.pv           binary(26) array           ,

    c.lb           integer                    ,
    c.bcid         integer                    ,
    c.lpsk         integer                    ,
    c.etime        timestamp                  ,
    c.id           bigint                     ,
    c.tbp          smallint array             ,
    c.tap          smallint array             ,
    c.tav          smallint array             ,

    d.lb1          integer,
    d.bcid1        integer,
    d.hpsk         integer,
    d.lph          smallint array,
    d.lpt          smallint array,
    d.lrs          smallint array,
    d.ph           smallint array,
    d.pt           smallint array,
    d.rs           smallint array,

    CONSTRAINT events_pk PRIMARY KEY
    (dspid, dstypeid, eventno, seq)
) DATA_BLOCK_ENCODING='FAST_DIFF', COMPRESSION='SNAPPY';
```

Data structures in HBase/Phoenix

Auxiliary tables to keep the dataset generated identifiers and bookkeeping data

- Datasets table:
  - Dataset location
  - Generated identifiers (dspid)
  - Import status
  - Some metadata information: number of all events, unique events, duplications
- Data types table:
  - data types (RAW, EVNT, AOD, DAOD, . . .)
  - subtypes for derivations

New system performance - Data ingestion

Data ingestion

A new back-end plugin for HBase/Phoenix ingestion → the Consumer part of the Data Collection architecture

- First test: inserting 48 2018 Tier-0 production datasets
- 3 kHz events/s per single-thread consumer

Operation	% of the time
data mangling	1
waiting for data	1.5
parsing and converting data from the old schema to the new	8.5
inserting into JDBC buffers	41
flushing the data committing buffers to Phoenix	48

New system performance - Data ingestion

Production data importer

use Map/Reduce or Spark jobs to import the existing production data from the Hadoop MapFiles

Tests:

- individual ingestion of datasets with sizes from 1 MB (500 Hz ingestion rate) to 10 GB (4.2 kHz)
- massive ingestion of 8000 datasets, containing 70 billion events:
  - 115 kHz mean ingestion rate
  - took one week
  - majority of datasets were imported correctly, without failures on input corrupted data or bad records, or failures converting to the new Phoenix schema

New system performance – Use cases

Queries to check the use cases were performed on a standalone test cluster, and on the Analytix production cluster

Queries performance of the HBase/Phoenix tables

	Query description	Timing
(1)	Counting canonical datasets (with the same project, run number, stream name, production step and version, but different data type) with their derivations	≈ 2 sec
(2)	Count events from a derivation (from the Datasets table)	≈ 27 msec
(3)	Get all events from a derivation (from the Events table)	≈ 2 min
(4)	Operations using trigger information: first get the dspid	
(5)	Operations using trigger information: use the dspid to get L1 trigger counts grouped by luminosity block	10-100 sec (depending on dataset size, including step (4))
(6)	Event lookup	< 1 sec

New system performance

- Queries done while writing data (1st batch of tests) take more time to complete.
- Some queries last much longer on the production cluster than on the test cluster. Possible reasons:
  - Different number of events (the test table was 20 times smaller)
  - Different table configuration
  - Lack of statistics, because they were not accessible (permissions) or not yet computed
  - Scans could be also affected by not having all compactions on the data
  - The first query on a table takes about 10 s as it loads the region location and statistics reduced if we go for bigger regions, since HBase is using guideposts when scanning
  - Parallelism can be probably increased by lowering the size of the guidepost and recomputing the statistics

Graph layer

- A graph database layer working on top of any SQL database has been implemented
- delivers interactive view of the EI data stored in the Phoenix SQL database
- All data are accessed directly via the standard Gremlin API and the interactive graphical Web Service

Graph view of ATLAS data

Conclusion

- Since the end of LHC Run 2 the current implementation of the EI (Hadoop MapFiles and Hbase) started showing scalability issues as the amount of stored data increases
  - Slower queries, lots of storage (now eased by compression)
- The significant increase in the data rates expected in future LHC runs demands transition to a new technology
- A new prototype based on Hbase+Phoenix system has been tested and shows encouraging results
  - good table schema was designed
  - the basic functionality is ready

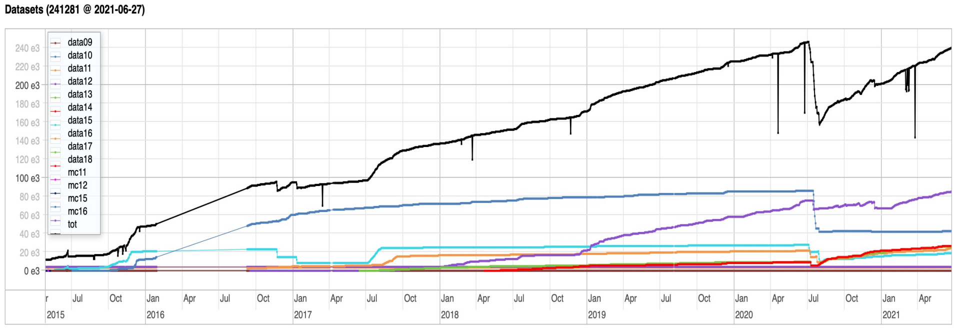
=> The ongoing work is aimed on performance improvement and interfaces development to have the system in operation well in advance of the start of Run 3 in 2022

Thank you for your attention!

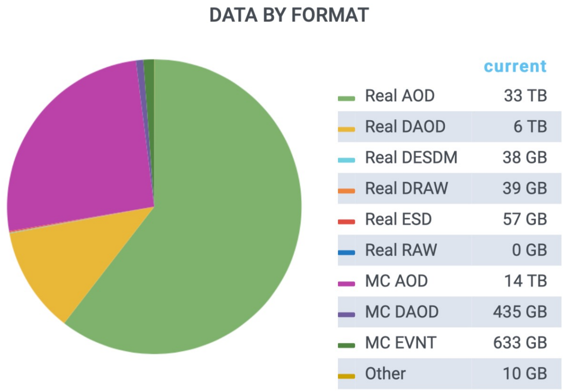
Back up

Operation and performance

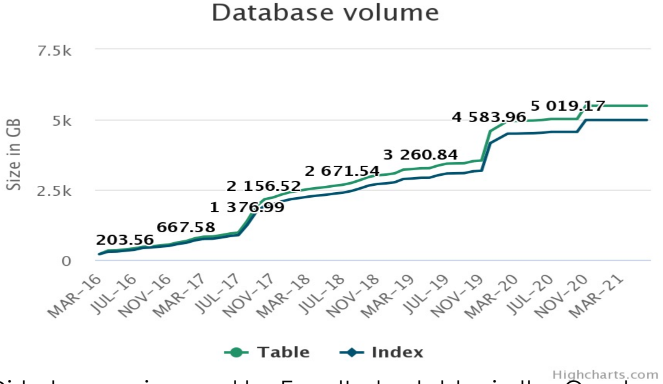
The Hadoop system runs a variety of tasks importing and cataloguing data, running consistency checks, establishing links between related datasets and responding to users queries



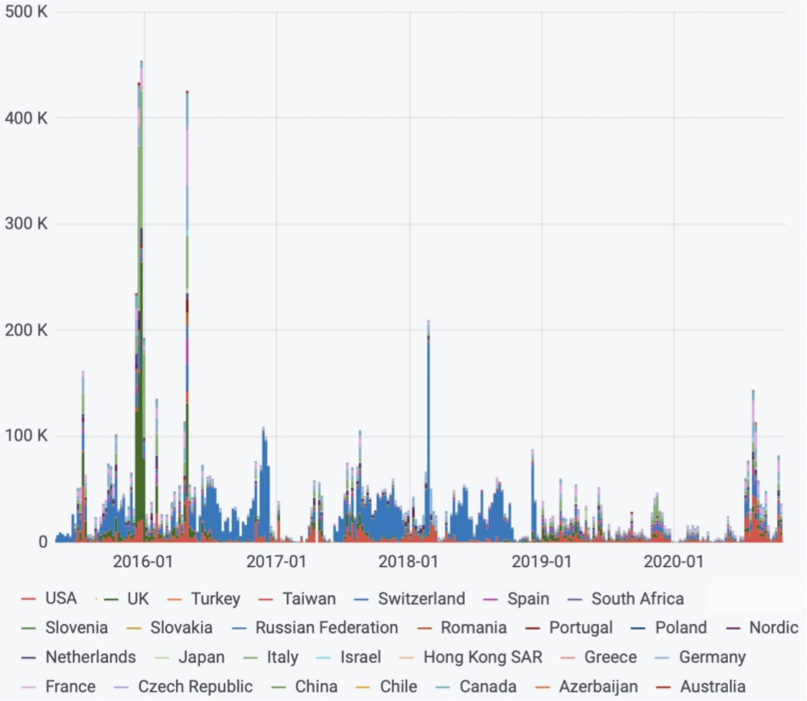
Datasets stored in the Hadoop system between May 2015 and June 2021



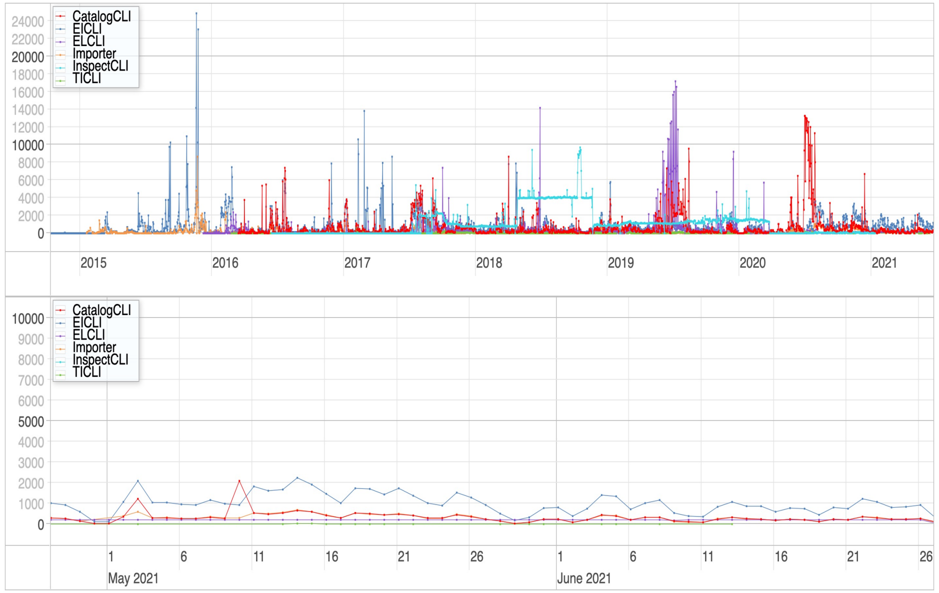
Data volume used in the Hadoop cluster, split by data type, June 2021



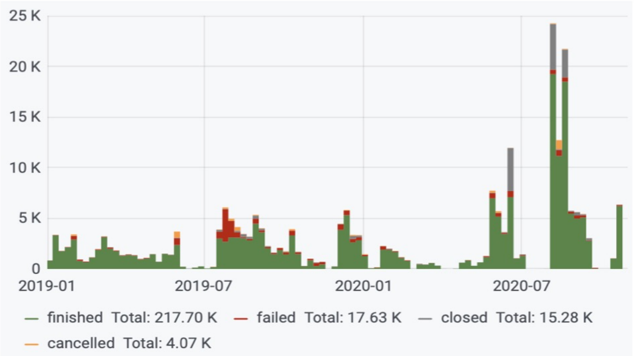
Disk storage size used by EventIndex tables in the Oracle cluster



Distribution of the EventIndex Producer jobs run each week world-wide between the start of operations in May 2015 and October 2020. Jobs run as close as possible to the in- put data location.



Access statistics of the Hadoop system between May 2015 and June 2021(top) and between Apr 2020 and Jun 2021 (bottom)



Distribution of event picking jobs run each week world-wide between January and October 2020

Trigger Counter

web service able to provide information about the trigger composition of a given dataset. It can:

- count the occurrences of each trigger in the dataset. The counts can be grouped by LumiBlock Number or bunch crossing identifier
- provide a list of events, given trigger-based search criteria
- calculate the overlap of the triggers in the dataset
- apply a trigger selections with filters

Search results are displayed in interactive plots, also in JSON or CSV format

Trigger Counter

The web form to use the Trigger Counter

Run Number: 341312

Stream: physics_BphysLS

Dataset: data17_13TeV.00341312.physics_BphysLS.merge.AOD.f903_m1917

Operation: COUNT

Level: HLT

Trigger: PH

Expression

Group by: No

Limit

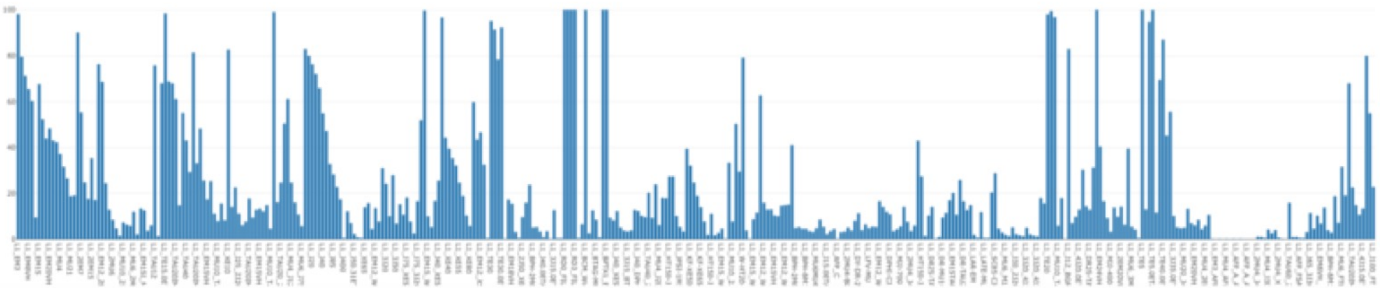
Run!

Show/Hide Triggers +

Results of f0456a8c-f830-40b1-b709-00da53a8bc03

COUNT TBP from data17_13TeV.00327103.physics_Main.merge.AOD.f832_m1812 where L1_EM3 or L1_MU21

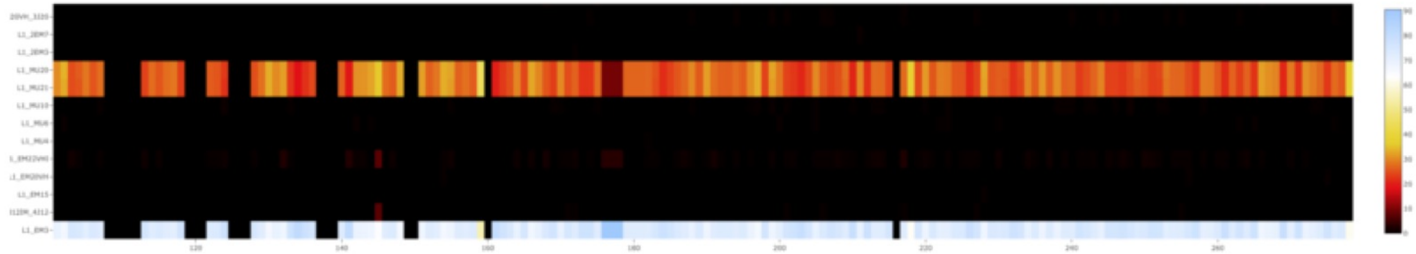
Example of a display of global trigger counts



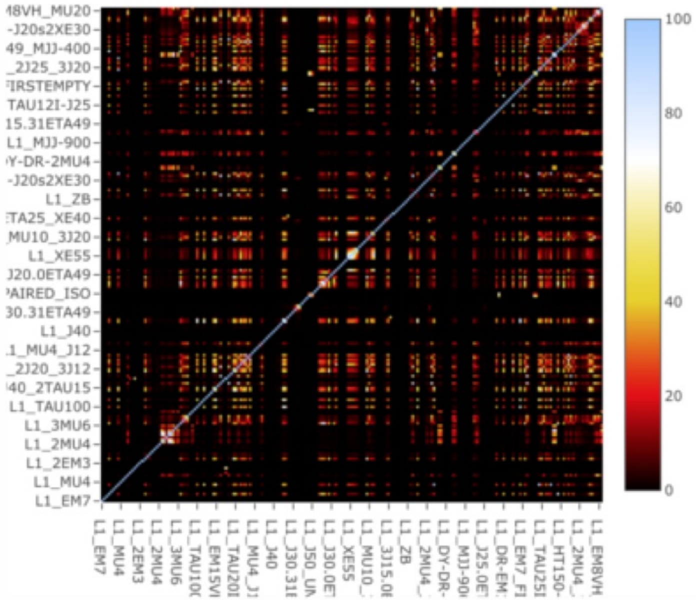
Results of 86cc7e49-e17a-4bb6-b524-2701441b5f61

COUNT TAV from data17_13TeV.00327103.physics_Main.merge.AOD.f832_m1812 where L1_EM3 or L1_MU21 group by LumiBlock

Example of a displays of counts grouped by luminosity block number



The heat map of trigger overlaps within a given dataset



Useful links

- EventIndex main Twiki page: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/EventIndex
- EventIndex tutorial: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/EventIndexTutorial
- Use cases: https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/EventIndexUseCases
- Monitoring Grafana dashboards: https://monit-grafana.cern.ch/d/RBMMmOnmz/atlas-eventindex?from=now-7d&orgId=6&to=now
- Contact developers: atlas-db-eventindex@cern.ch