Distributed Data Collection for the Next Generation ATLAS EventIndex project

Álvaro Fernández Casaní, Dario Barberis, Javier Sánchez, Carlos García Montoro, Santiago González de la Hoz, Jose Salt Cairols

on behalf of the ATLAS Collaboration

IFIC INSTITUT DE FÍSICA CORPUSCULAR

Instituto de Física Corpuscular (IFIC)
CSIC - Universitat de València

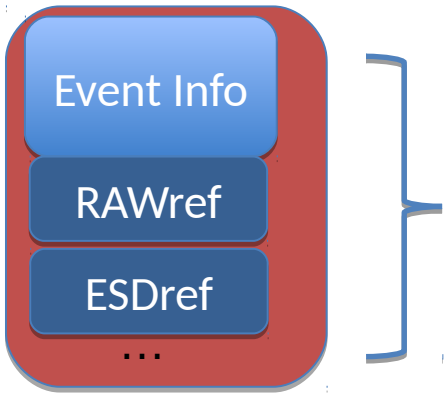
CHEP 2018
23RD INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS
9-13 July 2018
National Palace of Culture
Sofia, Bulgaria

Outline

1) ATLAS EventIndex project
    - Objective and use cases
2) Distributed Production and Data Collection Architecture
3) Performance and results
4) Evolution guidelines for Next Generation EventIndex
    - Testing Kudu as a new Backend Storage.
5) Summary

1. ATLAS EventIndex: an event catalog

- **A catalog of data** (all events in all processing stages) is needed to meet multiple use cases and search criteria. A small quantity of data per event is indexed.



Events stored in files (identified by GUID)
Files are grouped into DATASETS
Wanted Event Index information ~= 300bytes to 1Kbyte per event:

- Event identifiers (run / event numbers, trigger stream, luminosity block)
- Online trigger pattern (l1, l2, ef)
- References (pointers) to the events at each processing step (RAW, ESD, AOD, DAOD) in all permanent files on storage

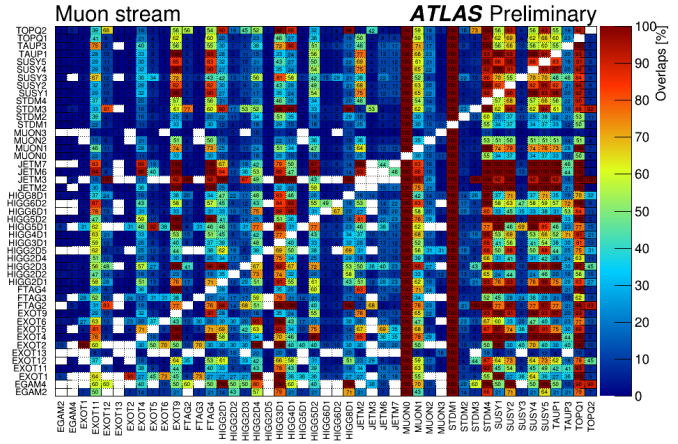
- We are indexing **Billions of Events,** stored in **Millions of files** replicated at CERN and **hundreds of grid-sites worldwide,** adding **100 Petabytes of data→ A complex big data distributed system.**

Use Cases

1) Event picking: users able to select single events depending on constraints. Order of hundreds of concurrent users, with requests ranging from 1 event (common case) to 30k events (occasional).

2) Production consistency checks

- **Duplicate event checkings**: events with same Id appearing in same or different files/datasets.
- **Overlap detection** in derivation framework: construct the overlap matrix identifying common events across the different files.

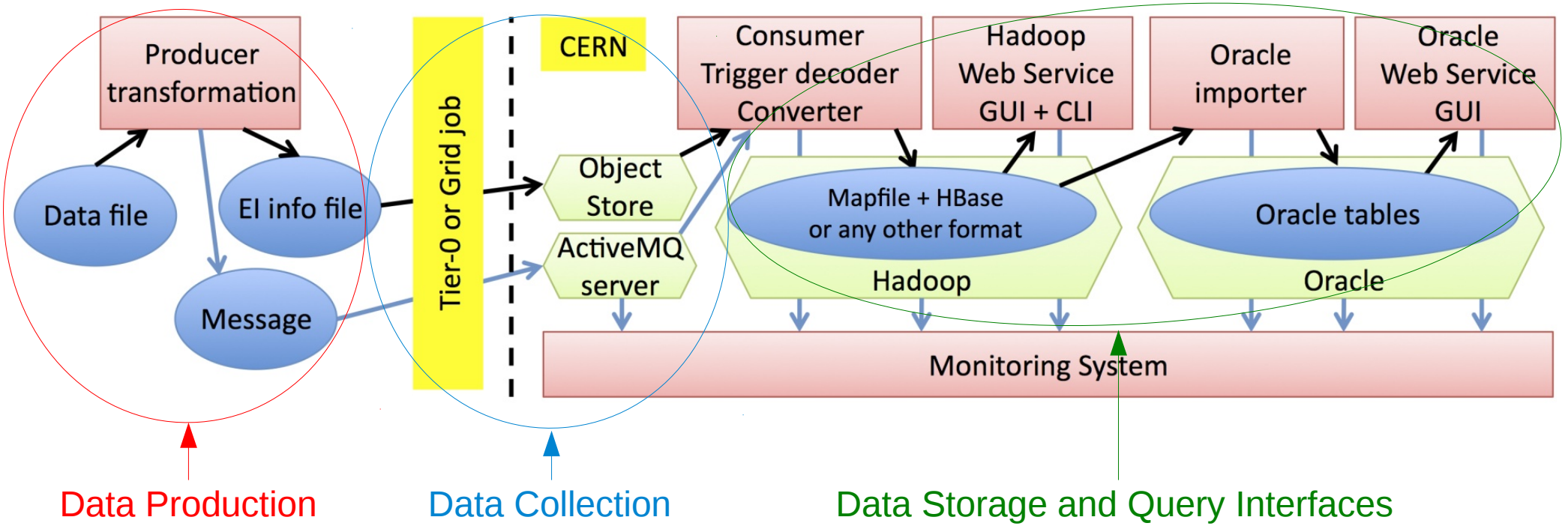


3) Trigger checks and event skimming: Count or give an event list based on trigger selection.

- **Trigger Overlap detection:** number of events in a real data Run/Stream satisfying trigger X which also satisfies trigger Y.

Requirement: Storing and accessing thousands of files and millions of events in reasonable time.

2. EventIndex Architecture

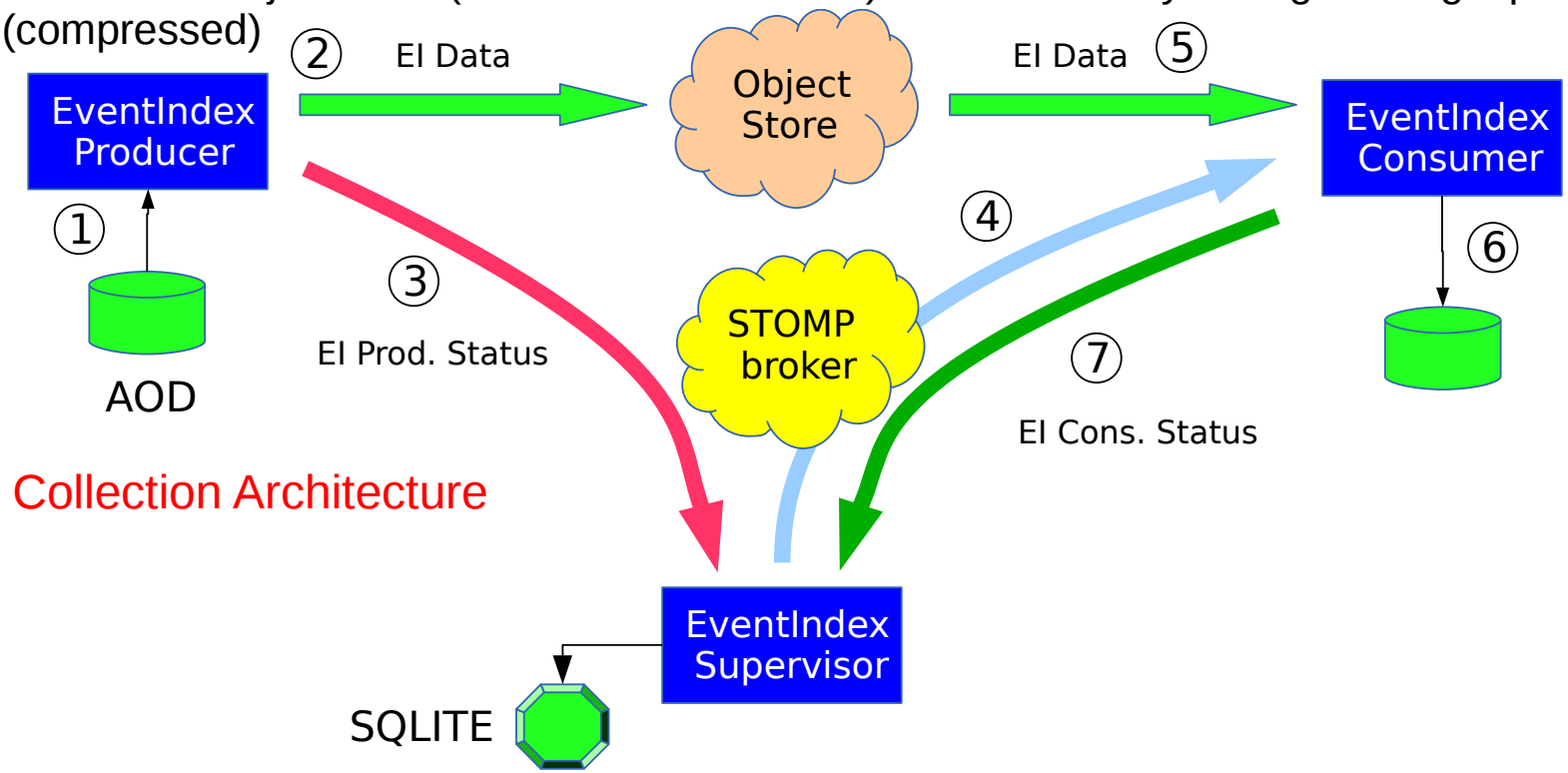


Data Production task to ensure all event index grid production performs correctly.

Data Collection task: a distributed producer/consumer architecture to collect all indexed data and ingest it to the Data Storage services.

Data collection

2015 - mid 2017: Pure Messaging Based architecture (ActiveMQ brokers / Stomp protocol). Json data encoding. (production peaks showed bottlenecks on messaging brokers)
mid 2017 onwards: ObjectStore (CEPH / S3 interface) as intermediary storage. Google protobuf data encoding (compressed)

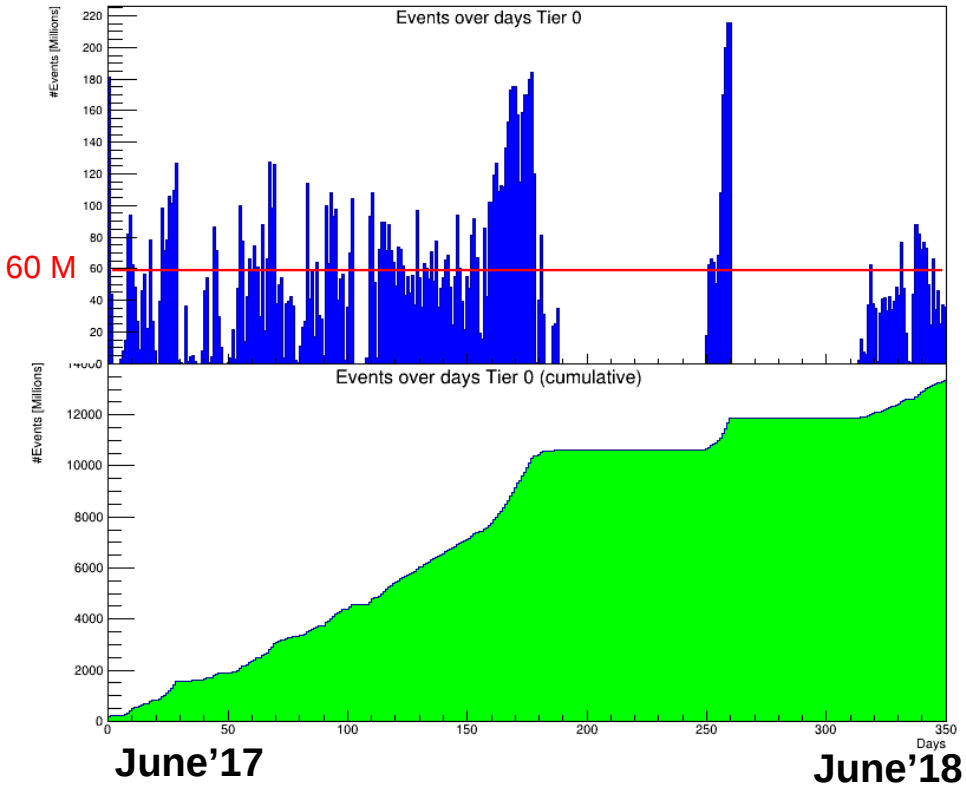


2018 Data Collection Architecture

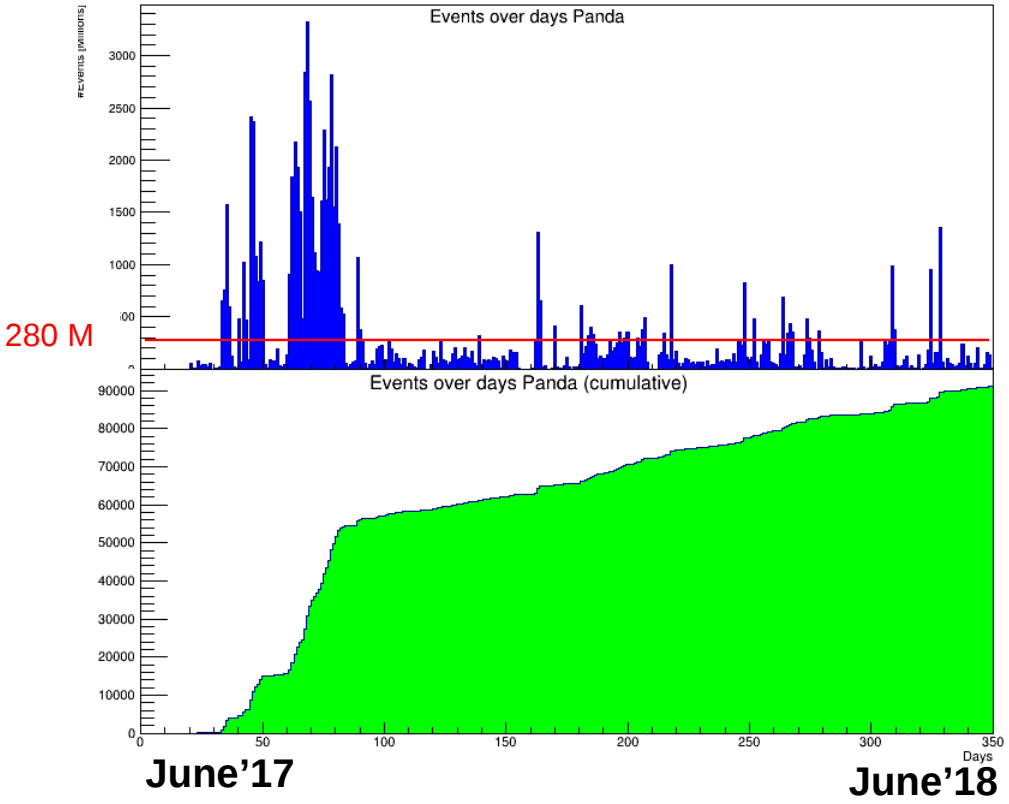
Producer: Athena Python transformation, running at Tier-0 and grid-sites. Indexes AOD data and produces an **eventindex file**, stored in **ObjectStore**
Supervisor: Controls all the process, receives processing information and validates data by dataset. Signals valid unique data for ingestion to Consumers. Operated with a web interface
Consumers: Retrieves ObjectStore data, groups by dataset and ingest it into **HDFS (Hadoop distributed Filesystem)**

3. Performance: Event processing rates

TIER0 @ CERN



GRID



EventIndex data production:

Tier0 @ CERN : ~60 M events/day

Grid Sites: ~280 M events/day

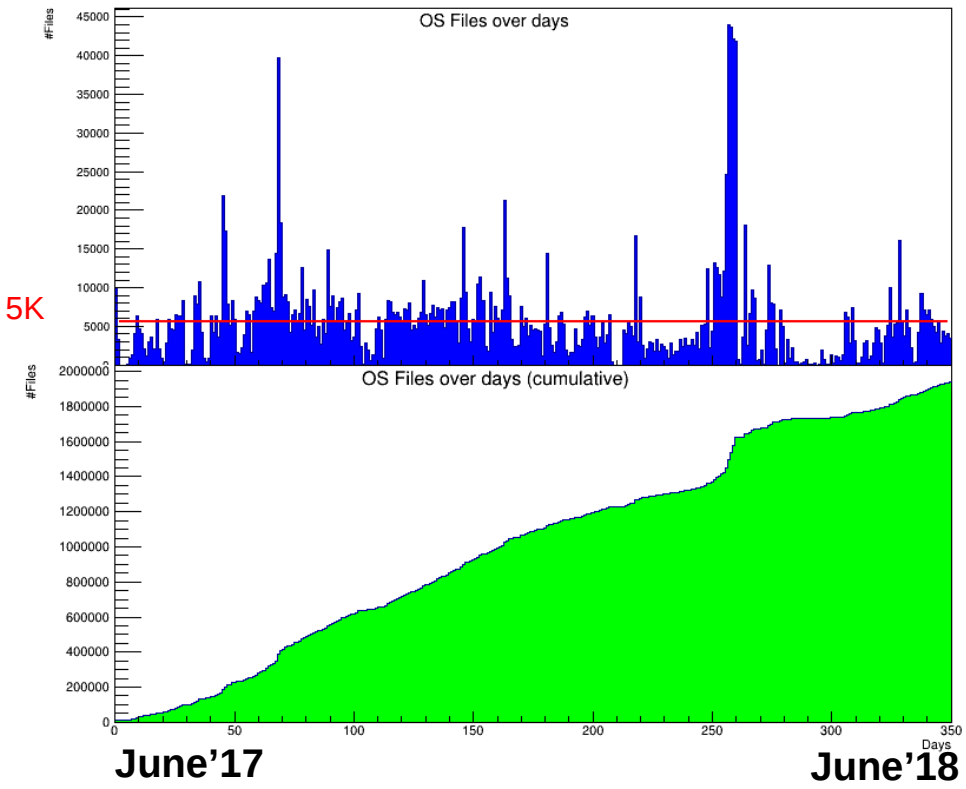
Total (Tier0 + Grid)

Peaks of 3500 M events indexed per day

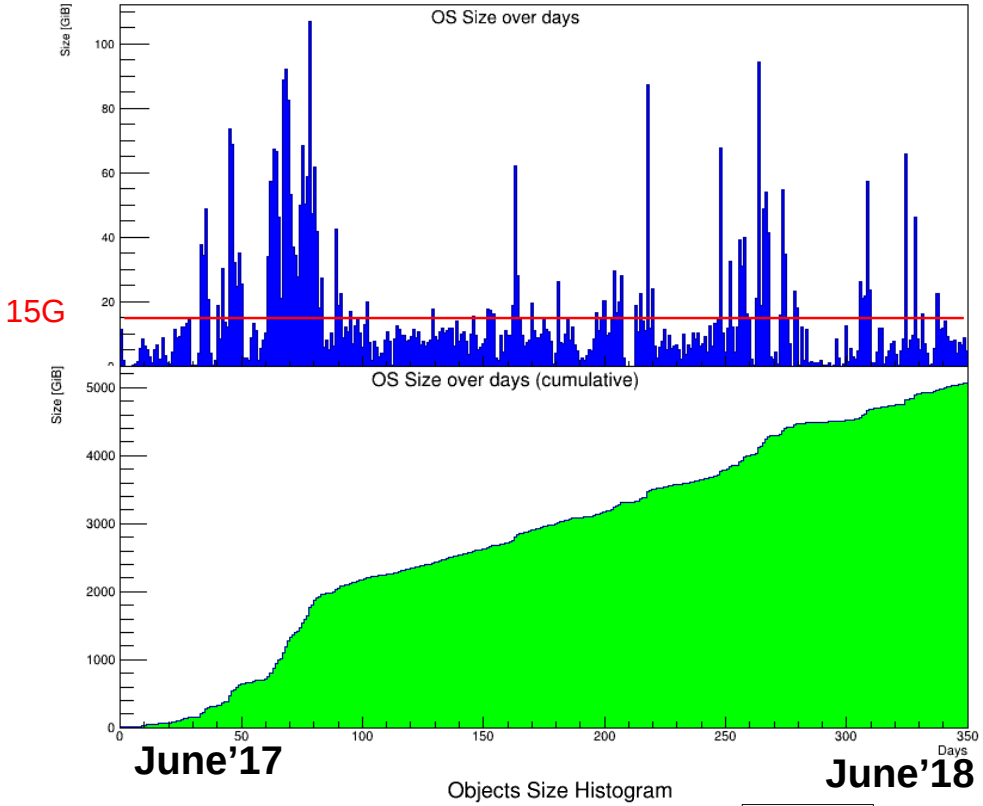
105 Billion events indexed during last year (350 days) with ObjectStore approach

Object Store rates

Object creation rates



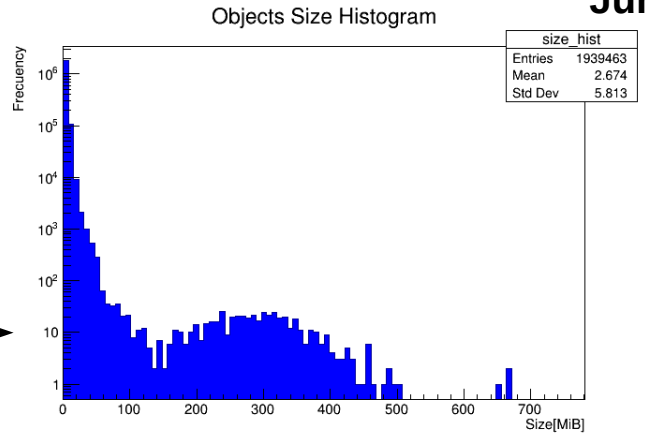
Size creation rates



Object creation rates:
- Mean of 5600 objects created/day with peaks of 44K objects.
- A total of 2M objects residing on CERN Ceph Object Store.

Size creation rates:
- Mean of 15 GiB stored/day, with peaks of 100GiB.
- A total of ~5 TiB stored on CERN Ceph Object Store.

Objects Size:
- Mean object size: 2.6 MiB (99% of objects are less that 15 MiB)
- Some bigger object, biggest one is 670 MiB
- Factor 10 compressed with respect to original eventindex data.

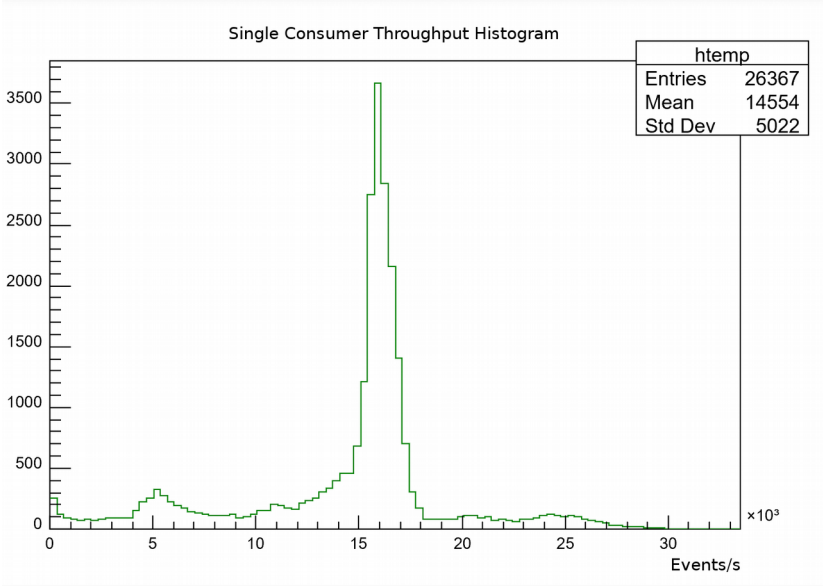


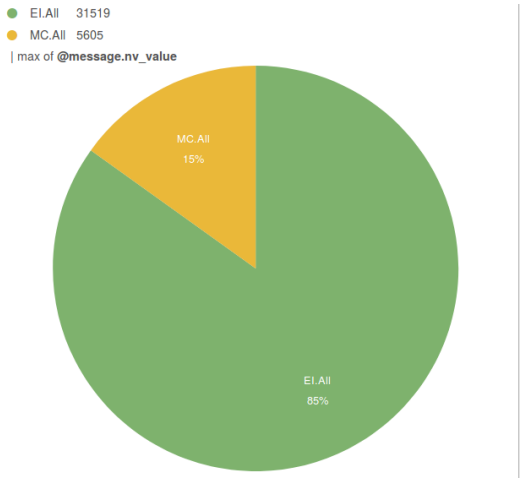
Data Ingestion to Hadoop HDFS

- **Consumers retrieve the eventindex files from ObjectStore and write it in HDFS Hadoop**
  - Granularity at the dataset(tid) level
    - 40% contained in a single object.
    - Most of the datasets < 75 MiB
    - Biggest one: 8000 objects (~7GiB)
  - Single Consumer event throughput performance improved from 1K events/s (Messaging only), to 15K events/s (ObjectStore).
  - Overcoming messaging brokers bottleneck, we can also now scale horizontally.
  - Stored in a single HDFS file per dataset (tid) in a directory named after the container. Reduced the number of HDFS files compared with previous approach with a HDFS file per indexed GUID.

- **Current (all years) EventIndex Data in Hadoop:**
  - 37 TiB of indexed events data (167 TiB before compression): 31 TiB real data, 6 TiB MonteCarlo simulated data

Single Consumer rates





Hadoop (HDFS)

4. Evolution guidelines for Next Generation EventIndex

- **An evolution of the EventIndex concepts**
  - **Currently**: the same event across each processing step (RAW, ESD, AOD, DAOD, NTUP) is physically stored at different HADOOP HDFS files.
  - **Future**: One and only one logical record per event (Event Identification, Inmutable information (trigger, lumiblock, …), and for each processing step:
    - Link to algorithm (processing task configuration)
    - Pointer(s) to output(s)
    - Flags for offline selections (derivations)
- **Support Virtual Datasets:**
  - A logical collection of events
    - Created either explicitly (giving a collection of Event Ids) of implicitly (selection based on some other collection or event attributes)
    - Labelling individual events by a process or a user with attributes (key:value)
- **Evolve EventIndex technologies to future demanding rates:**
  - **Currently:** ALL ATLAS processes: ~30billion events/day (up to 350Hz on average) → update rate throughout the whole system (all years, real and simulated data). Read 8 M files/day and produce 3 M files
  - **Future:** due to expected trigger rates, need to scale for next ATLAS runs: at least half an order of magnitude for Run3 (2021-2023): 35 B new real events/year and 100 B new MC event/year. For run4: 100 B new real events and 300 B new MC events per year. Then sum up replicas and reprocessing

Exploring new backend storage solutions

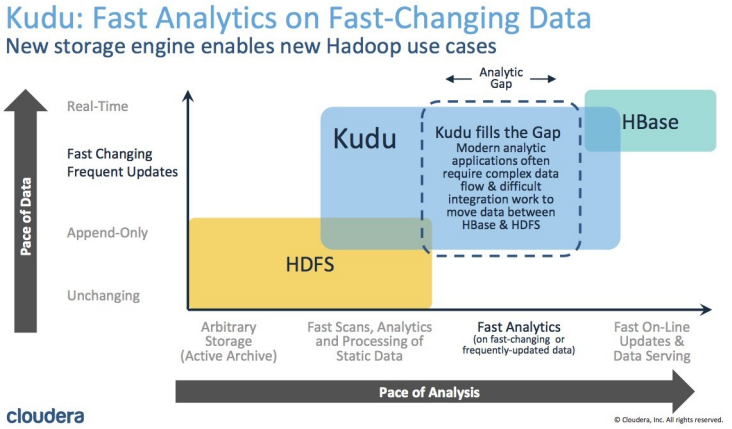
Optimization and unification of data storage for the EventIndex

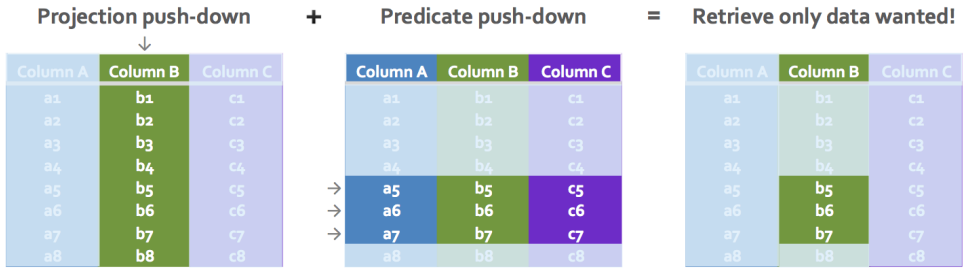
Apache Kudu: new columnar-based storage that allows fast insertions and retrieval.

- Tables with defined schema, primary keys and partitions. No foreign keys
- Ingestion and query scanning are distributed among the servers holding the partitions (tablets)
- Partition pruning and projection/predicate pushdown

Benefits for EventIndex:

- Unify data for all use cases (random access + analytics)
- Related data (reprocessings) sit close to each other on disc. Reduce redundancies and improve navigation.





- See poster in this conference for more information: “A PROTOTYPE FOR THE EVOLUTION OF ATLAS EVENTINDEX BASED ON APACHE KUDU STORAGE”

Kudu events table schema

KEY columns	
<epoch, project, streamname, prodstep, datatype, version, runnumber, eventnumber>	• Event uniqueness is forced among all tids and files • Key keeps events that belong to same dataset close • Scans for events in the same dataset are concentrated • Horizontal Partitions by HASH(eventnumber) and RANGE(epoch) on Key columns.
Trigger columns -binary-	
L1: **24 columns x int64** = 1536 bits 8x(before prescaler), 8x(after prescaler), 8x(after veto) **HLT**: **192 columns x int64** = 12288 bits 64x(physics), 64x(Passthrough), 64x Resurrected	• To allow selection on individual triggers • Directly mapped to IMPALA bitwise operations (limited to 64bit operands) • Projection and predicate: push-down omitting unnecessary fields from table scan.
Trigger columns -text-	
L1mask: string (JSON) **EFmask:** string (JSON)	• JSON encoded trigger bits to allow versatile operations. Less columns that binary approach, but without the possibility of bitwise IMPALA operations.
Event location	
db, oid1, oid2	• GUID of the file where to find this event, and object id inside that file
provenance	• JSON. Where to find this event in previous processing versions
Event info	
Lumiblockn,bunchid,eventtime, eventtimens, lvl1id, hltpsk, l1psk	• Event info specific data. Information shared by dataset would go in another table.

Kudu Test setup

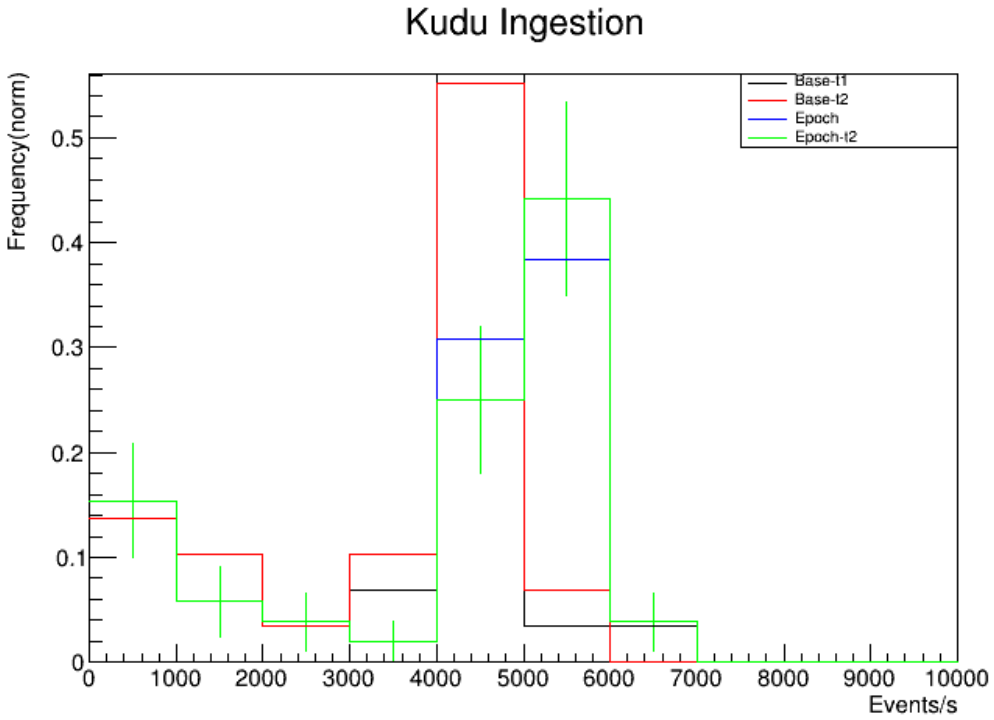
- Current setup at IFIC
  - Kudu 1.7 + Impala 2.11 +Spark 1.6 (cdh5.14.2)
- 5 machines with:
  - 2x Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz (14 cores/CPU)
  - 16x 16 GB RAM DDR4 @ 2400 MHz (256 GB)
  - 8x data disks SATA SEAGATE ST6000NM0034 (6TB)
  - 1x os disk SSD SAMSUNG MZ7KM240 (240GB)
  - 1x Intel SSD DC P3700 (1.5 TB) pci nvme
  - 2x 10Gpbs ethernet controller
- Current configuration:
  - 1 master, 4 tablet servers
  - 1 big data disk (RAID10) to store tablets (22TB per machine)
  - WAL on Intel SSD

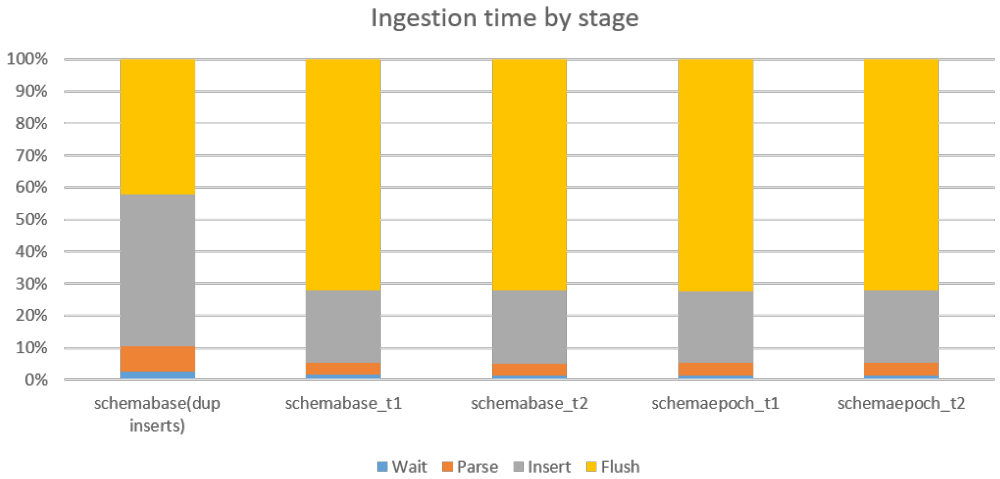
Trigger encoding test

- Data in KUDU
  - L1: 8x(before prescaler), 8x(after prescaler), 8x(after veto) — **24 columns x int64** = 1536 bits
  - HLT: 64x(physics), 64x(Passthrough), 64x Resurrected — **192 columns x int64** = 12288 bits
  - L1mask: string (JSON)
  - EFmask: string (JSON)

	Columns total size in Bytes/event		
trig	BIT_SHUFFLE LZ4 0 as null	PLAIN_ENCODING LZ4 0 as null	PLAIN_ENCODING LZ4 0 as 0
L1	59.96	65.17	69.28
HLT	14.97	26.96	45.16
mask	DICT NO_COMPRESSION	PLAIN_ENCODING LZ4	PLAIN_ENCODING LZ4
L1Mask	697.55	150.93	150.93
EFMask	680.86	45.47	45.47

Kudu ingestion test results





- 1 consumer per table performance

- Input data: datasets from May 2018 (mainly tier0)
- Tested different tables/configuration:

Base-t1: HASH(eventnumber)=8 RANGE(runnumber)
-all May'18 ds in same range(runnumber)
Base-t2: same as Base-t1 with key ending <…,runnumber, eventnumber>
Epoch: HASH(eventnumber)=4 RANGE(epoch)=4
Epoch-t2: HASH(eventnumber)=8 RANGE(epoch)=4

Ingestion mean rate: ~5K events/s

Consumer Ingestion Stages:
Wait: for data valid (1%)
Parse: data conversion (4%)
Insert: into Kudu client buffers (23%)
Flush: buffers to Kudu (72%)

Summary

- EventIndex Distributed Data Collection is running in production indexing and collecting billions of events worldwide. During last year we have indexed 300 M events per day.
- Object Store based improved previous messaging-only approach:
  - Producer payload encoded in a single object.
  - More compact binary data encoding using protocol buffers. Compression reduces a factor 10 the data.
  - Supervisor selects unique validated data, without consuming duplicate data into HDFS.
  - Consumer improved performance. No blockings detected, we can scale horizontally adding new instances when needed.
- Future challenges regarding new production rates, and EventIndex use cases evolution:
  - Current work on new Storage Technologies to support faster insertion, and random access (low latency) and analytics use cases unification.
  - First Ingestion tests on Kudu shows a promising backend for the EventIndex project.
  - See poster in this conference for more information: “A PROTOTYPE FOR THE EVOLUTION OF ATLAS EVENTINDEX BASED ON APACHE KUDU STORAGE”