The ATLAS EventIndex for LHC Run 3

Dario Barberis[1,*], *Igor* Aleksandrov[2], *Evgeny* Alexandrov[2], *Zbigniew* Baranowski[3], *Gancho* Dimitrov[3], *Álvaro* Fernández Casaní[4], *Elizabeth J.* Gallas[5], *Carlos* García Montoro[4], *Santiago* González de la Hoz[4], *Julius* Hrivnac[6], *Andrei* Kazymov[2], *Mikhail* Mineev[2], *Fedor* Prokoshin[2], *Grigori* Rybkin[6], *Javier* Sánchez[4], *José* Salt Cairols[4], and *Miguel* Villaplana Perez[7]

[1]Physics Department of the University of Genoa and INFN Sezione di Genova, Via Dodecaneso 33, I-16146 Genova, Italy
[2]Joint Institute for Nuclear Research, 6 Joliot-Curie St., Dubna, Moscow Region, 141980, Russia
[3]CERN, 1211 Geneva 23, Switzerland
[4]Instituto de Fisica Corpuscular (IFIC), Centro Mixto Universidad de Valencia - CSIC, Valencia, Spain
[5]Department of Physics, Oxford University, Oxford, United Kingdom
[6]Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France
[7]Department of Physics, University of Alberta, Edmonton AB, Canada

Abstract. The ATLAS EventIndex was designed in 2012-2013 to provide a global event catalogue and limited event-level metadata for ATLAS analysis groups and users during the LHC Run 2 (2015-2018). It provides a good and reliable service for the initial use cases (mainly event picking) and several additional ones, such as production consistency checks, duplicate event detection and measurements of the overlaps of trigger chains and derivation datasets. The LHC Run 3, starting in 2021, will see increased data-taking and simulation production rates, with which the current infrastructure would still cope but may be stretched to its limits by the end of Run 3. This proceeding describes the implementation of a new core storage service that will be able to provide at least the same functionality as the current one for increased data ingestion and search rates, and with increasing volumes of stored data. It is based on a set of HBase tables, with schemas derived from the current Oracle implementation, coupled to Apache Phoenix for data access; in this way we will add to the advantages of a BigData based storage system the possibility of SQL as well as NoSQL data access, allowing to re-use most of the existing code for metadata integration.

1 Introduction

The ATLAS experiment [1] at the LHC accelerator at CERN collected during the so-called "Run 2" (2015-2018) several billion physics events each year, plus a large amount of test and calibration data. In addition, about three times as many fully simulated events were produced, stored and analysed together with the real collision data. A global catalogue is

[*] Corresponding author: Dario.Barberis@cern.ch
© 2020 CERN for the benefit of the ATLAS Collaboration. CC-BY-4.0 license.

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

needed to classify, search and retrieve events from this vast amount of data: the EventIndex. Its design started in 2012-2013 [2] by exploring the new technologies that became available at that time in the open-source software community, then called "NoSQL databases". The most promising solution was based on the Hadoop eco-system [3], with data stored in HDFS MapFiles and an internal catalogue in HBase. This system was pre-loaded with all Run 1 real data (2009-2013) and started operation in the Spring of 2015 at the start of Run 2 [4]. It was the first high-energy physics computing system based from the start on open-source structured storage technologies.

2 The EventIndex for LHC Run 2

2.1 Use Cases

The main use case for the EventIndex is event picking. Collaboration members analysing particular datasets have the need from time to time to extract one particular event in an upstream data format to check if the reconstruction algorithms worked correctly, or to produce an event display to be used in a presentation or publication. Given the minimal unique information on the event, *i.e*. the run and event numbers, and the data format and version to be retrieved, the EventIndex has to provide the pointer to the file containing this event and the associated tools must be able to extract it from this file and return it to the requestor.

Additional use cases are related to production completeness and correctness checks. At every processing step, the output files must contain the same number of events as the inputs, and there must be no duplicated event numbers. The ability to count events in files or datasets depending on some internal information, like triggers that are fired, is also an important use case.

As the data actually became available in the EventIndex, two more use cases joined the list during Run 2: studying trigger overlaps, and studying the overlaps between offline analysis derivation streams, with the aim of optimising the event selections both online and offline.

All the above use cases are satisfied by indexing all produced data, as soon as each dataset is completed, and storing the event information in the EventIndex data store. A side-effect of this operation mode is the global check for corruption of all produced data, as in this way each file is read back by an independent process soon after being produced, and before being used for further processing or analysis.

2.2 Data Contents

ATLAS event data are organised into datasets, which are collections of files that contain similar numbers of events from the same run or period, or simulated events created with the same generator settings, which have been processed with the same software algorithms and calibrations as the real data.

Each dataset is indexed as soon as it is produced, therefore the EventIndex contains event data organised by dataset. Only immutable technical data ("metadata"), as opposed to physics information that can change at each processing stage, are saved in the EventIndex:

- Event identification data: run and event number, luminosity block number, bunch crossing identifier;
- Trigger decisions;

- References to the events at each processing stage in all permanent files generated by central productions: globally unique identifiers (GUIDs) of the files containing the event at the current processing stage and previous ones if available.

The last block of information is needed for event picking. The GUID retrieved by the query to the EventIndex catalogue is used to find the relevant file in the Grid data store and extract the requested event.

2.3 System Architecture

Since the start of the project, it was clear that a partitioned architecture, following the data flow across the system, would have been the best solution, and indeed this was the case. A schematic illustration of the system architecture is shown in Figure 1.

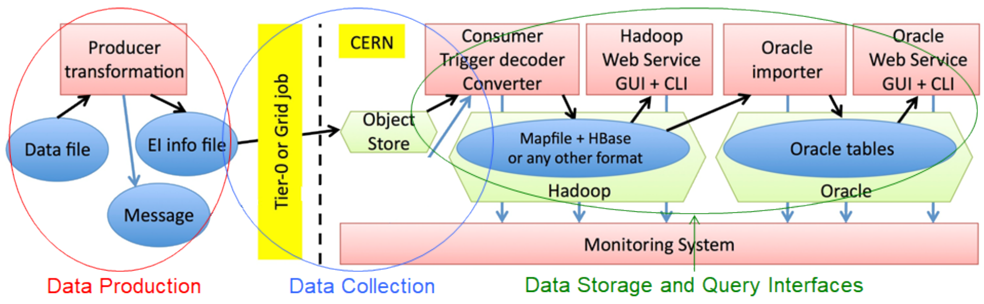


Fig. 1: System architecture of the EventIndex, showing the main components and the data flow.

The *Data Production* sub-system automatically runs jobs on the Tier-0 farm or on the Grid on each dataset as soon as it is produced and extracts the metadata to be stored in the EventIndex.

The *Data Collection* component [5] takes care of transferring the EventIndex metadata from the production sites to CERN, merging it for all files of each dataset, validating it and storing the result in temporary files on the Hadoop cluster.

The *Data Storage and Query* system was initially implemented purely in Hadoop, with the data in the internally indexed MapFile representation and a dataset catalogue implemented in HBase [6]. The data flow in the Hadoop cluster includes reading the temporary files produced by the Data Collection step, expanding the compressed trigger information [7], storing the data in MapFiles and updating the HBase catalogue. However, it was soon realised that the overhead that is intrinsic in the start of MapReduce jobs in the Hadoop cluster was too large for simple operations such as finding an event record for the event picking use case, therefore an additional data store implemented in HBase was added. It contains all data except the trigger information, which constitutes the majority of the data volume; only queries needing this information fire up MapReduce jobs.

As the performance in retrieving information from the Hadoop system was still sub-optimal at the end of 2015, it was decided to add an alternative store, based on an Oracle database, to support a well-defined set of queries [8]. All real events now have their metadata also stored in an Oracle database, albeit without trigger information. This data store turned out to be faster and more reliable for event-picking related queries and is also useful for the integration between event-level metadata and other metadata that are also stored in Oracle databases.

At the end of 2019 the EventIndex used 24 TB to store 2009-2019 real data and 11 TB for 2015-2019 simulated data in Hadoop, and 3.4 TB of table space plus 3.1 TB of auxiliary index in Oracle (only 2015-2019 real data without trigger information).

Finally, the *Monitoring System* [9] provides real-time information on the status of the servers and services that compose the EventIndex systems and sends alarms to the experts in case of suspected problems.

3 The EventIndex for LHC Run 3

The current EventIndex storage implementation reflects the state of the art for BigData storage in 2012-2013 when the project started, but many different options have appeared since then, even within the Hadoop eco-system. The current system works well for the current injection and query rates, but with the increase of data-taking and simulation production rates foreseen for Run 3 (2021-2024) and even more for Run 4 (High-Luminosity LHC, 2027 onwards) it is clear that a re-design of the core systems is needed. In order to be safe, a new system should be able to absorb a factor 10 more event rate than the current one, *i.e.* 100 billion real events and 300 billion simulated events each year.

3.1 New Requirements

Nowadays the same event is physically completely stored in different Hadoop MapFiles for each data format and processing version. In the future we would like to have one and only one logical record per event, independently of the format the events are stored in. The base record for each event should contain only the event identifiers and the immutable information (trigger, luminosity block and bunch crossing number), and then for each processing step an additional record would contain the link to the algorithm that produced it (processing task configuration), the pointer(s) (GUIDs or equivalent) to the output(s), and any useful flags for offline selections (derivations).

A few additional use cases became important in the last few years:

- Massive event picking: this is a selection of many events, touching a large fraction (or perhaps all) of the files in a dataset. It will need a dedicated service, especially if the input files are on tape (which is normally the case for raw data);
- Adding "offline trigger" information: it entails storing the results of selections that can be used to form derived datasets. Related to this, there is the need for the ability to add information to parts of the event records, to have the possibility to select events using online and offline trigger information to build "virtual datasets";
- Support for virtual datasets: they are logical collection of events created either explicitly (giving a collection of event identifiers) or implicitly (their selection is based on some other collection or event attributes).
- Labelling individual events with attributes such as "key:value" pairs.

3.2 System Design Evolution

The global architecture supports the independent evolution of the system components, and indeed some of them have already been substantially improved or replaced by new implementations.

The Data Collection system was progressively restructured with the replacement of the messaging system ActiveMQ [10] with the CEPH Object Store [11] as the main data transfer mechanism between Grid jobs and the CERN central servers [12]. An EventIndex Supervisor was introduced at the same time to keep track of the data transfers, the validation procedures and the storage in Hadoop.

The Monitoring System was improved [13] with the replacement of the displays based on Kibana [14] with Grafana [15], following the trend set for other distributed computing applications supported by CERN for the ATLAS experiment [16].

Investigations on several structured storage formats for the main EventIndex data to replace the Hadoop MapFiles started a few years ago [17]. Initially it looked like Apache Kudu [18] would be a good solution, joining BigData storage performance with SQL query capabilities [19]. Unfortunately Kudu did not get a large support in the open-source community and CERN decided not to invest hardware and manpower resources in this technology.

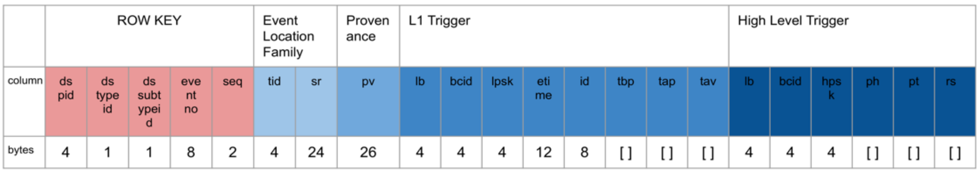
HBase was evaluated as the main data store at the beginning of the project, but discarded at that time because of performance restrictions. Nowadays it is able to hold the large amounts of data to be recorded, with a much-improved data ingestion and query performance thanks to the increased parallelisation of all operations. Additional components like Apache Phoenix can provide SQL access to HBase tables, if the tables are designed appropriately upfront, which can be done in our case.

3.3 Data Store in HBase+Phoenix

Data in HBase can be organised in large tables, with one row per event and one column per event property. Each row has to be identified by a unique key (RowKey), basically something like RunNumber-EventNumber in our case. Columns are grouped in column families, which can match easily our requirement of having groups of values that are added to each record at each processing stage.

Phoenix is used as the SQL layer on top of HBase. It provides structured schemas for the tables instead of the native HBase schema-less tables, the mapping of columns to HBase cells, and the serialisation of data types to bytes. It also contains a SQL planner and optimiser with built-in HBase related optimisations, server-side optimised executions and access via JDBC (Java DataBase Connectivity). Phoenix takes SQL queries, transforms them into HBase commands using directly the HBase API, along with coprocessors and custom filters, and finally produces regular JDBC result sets.

The HBase RowKey design must be adapted to Phoenix's types and sizes, losing some performance but gaining functionality. Figure 2 shows a schema of the current RowKey design.



	ROW KEY					Event Location Family		Proven ance	L1 Trigger								High Level Trigger					
column	ds pid	ds type id	ds subt ypei d	eve nt no	seq	tid	sr	pv	lb	bcid	lpsk	eti me	id	tbp	tap	tav	lb	bcid	hps k	ph	pt	rs
bytes	4	1	1	8	2	4	24	26	4	4	4	12	8	[]	[]	[]	4	4	4	[]	[]	[]

Fig. 2: RowKey schema for the HBase"Phoenix version of the EventIndex data store.

A number of tests have been performed, loading ATLAS EventIndex data to HBase via Phoenix and then running Phoenix queries on the loaded data. The results are encouraging: single event picking works in 30 ms and full dataset queries run in 6-10 seconds. More tests and optimisation work are in progress. Some basic functions are ready and further work on performance and user interfaces is ongoing.

4 Conclusions and Outlook

The significant increase in the data rates expected in LHC Run 3 and the subsequent HL-LHC runs demanded the transition now to a new technology for the main EventIndex data store. A new prototype based on HBase event tables and queries through Apache Phoenix has been tested and shows encouraging results. There is a good table schema candidate and the basic functionality is ready. We are now working towards improved performance and better interfaces; of course, we need to keep testing the new system with more data and get performance metrics to compare with the current implementation.

The plan is to have the new system operational by the middle of 2020 in parallel with the current one and phase out the old system well in advance of the start of the LHC Run 3. According to our expectations, this system will be able to withstand the data production rates expected for LHC Run 4 and beyond.

References

[1] ATLAS Collaboration, JINST, **3**, S08003 (2008)
DOI: 10.1088/1748-0221/3/08/S08003.

[2] D. Barberis et al., J. Phys. Conf. Ser. **513** 042002 (2014)
DOI: 10.1088/1742-6596/513/4/042002.

[3] Hadoop, HBase and related tools: https://hadoop.apache.org.

[4] D. Barberis et al., J. Phys. Conf. Ser. **664** no. **4** 042003 (2015)
DOI: 10.1088/1742-6596/664/4/042003.

[5] J. Sánchez et al., J. Phys. Conf. Ser. **664** no. **4** 042046 (2015)
DOI: 10.1088/1742-6596/664/4/042046.

[6] A. Favareto et al., Phys. Part. Nucl. Lett. **13** no. **5** 621-624 (2016)
DOI: 10.1134/S1547477116050198.

[7] M. Mineev et al., CEUR Workshop Proceedings **2267** 104-107 (2018)
urn:nbn:de:0074-2267-5 http://ceur-ws.org/Vol-2267/104-107-paper-18.pdf.

[8] E.J. Gallas et al., J. Phys. Conf. Ser. **898** no. **4** 042033 (2017)
DOI: 10.1088/1742-6596/898/4/042033.

[9] D. Barberis et al., J. Phys. Conf. Ser. **762** no. **1** 012004 (2016)
DOI: 10.1088/1742-6596/762/1/012004.

[10] ActiveMQ: https://activemq.apache.org.

[11] CEPH: https://ceph.io.

[12] Á. Fernández Casaní et al., EPJ Web Conf. **214** 04010 (2019)
DOI: 10.1051/epjconf/201921404010.

[13] E. Alexandrov et al., CEUR Workshop Proceedings **2267** 91-94 (2018)
urn:nbn:de:0074-2267-5 http://ceur-ws.org/Vol-2267/91-94-paper-15.pdf.

[14] Kibana: https://www.elastic.co/products/kibana.

[15] Grafana: https://grafana.com.

[16] D. Barberis et al., CEUR Workshop Proceedings **2507** 23-29 (2019)
urn:nbn:de:0074-2507-4 http://ceur-ws.org/Vol-2507/23-29-paper-4.pdf.

[17] Z. Baranowski et al., J. Phys. Conf. Ser. **898** no. **6** 062020 (2017)
DOI: 10.1088/1742-6596/898/6/062020.

[18] Apache Kudu: https://kudu.apache.org.

[19] Z. Baranowski et al., EPJ Web Conf. **214** 04057 (2019)
DOI: 10.1051/epjconf/201921404057.