The ATLAS EventIndex and its evolution towards Run 3

M Villaplana Perez[1], E Alexandrov[2], I Aleksandrov[2], Z Baranowski[3], D Barberis[4], G Dimitrov[5], A Fernandez Casani[6], E Gallas[7], C Garcia Montoro[6], S Gonzalez de la Hoz[6], J Hrivnac[8], I Alexander[2], A Kazymov[2], M Mineev[2], F Prokoshin[2,9], G Rybkin[8], J Sanchez[6], J Salt[6] and P T Vasileva[3].

[1] INFN Sezione di Milano; Dipartimento di Fisica, Università di Milano, Milano, Italy
[2] Joint Institute for Nuclear Research, Dubna, Russia
[3] CERN-IT, Geneva, Switzerland
[4] INFN Sezione di Genova; Dipartimento di Fisica, Università di Genova, Genova, Italy
[5] CERN-ATLAS, Geneva, Switzerland
[6] Instituto de Física Corpuscular (IFIC), Centro Mixto Universidad de Valencia - CSIC, Valencia, Spain
[7] Department of Physics, Oxford University, Oxford, United Kingdom
[8] LAL, Universite Paris-Sud, CNRS/IN2P3, Universite Paris-Saclay, Orsay, France
[9] Departamento de Física, Universidad Técnica Federico Santa María, Valparaíso, Chile

E-mail: Miguel.Villaplana.Perez@cern.ch

Abstract. The ATLAS experiment has produced hundreds of petabytes of data and expects to have one order of magnitude more in the future. This data are spread among hundreds of computing Grid sites around the world. The EventIndex is the complete catalogue of all ATLAS events, real and simulated, keeping the references to all permanent files that contain a given event in any processing stage. It provides the means to select and access event data in the ATLAS distributed storage system, and provides support for completeness and consistency checks and trigger and offline selection overlap studies. The EventIndex employs various data handling technologies like Hadoop and Oracle databases, and it is integrated with other parts of the ATLAS distributed computing infrastructure, including systems for data, metadata, and production management. The project has been in operation since the start of LHC Run 2 in 2015, and it is in permanent development in order to satisfy the production and analysis demands and follow technology evolution. The main data store in Hadoop, based on MapFiles and HBase, has worked well during Run 2 but new solutions are being explored for the future. This paper reports on the current system performance and on the studies of a new data storage prototype that can carry the EventIndex through Run 3.

1. Introduction

The ATLAS experiment [1] produces several billion events per year. In order to retrieve the selected events from data storage systems, a database is necessary, with the references to the files including each event in every stage of processing. The ATLAS EventIndex project [2, 3] provides a way to store the necessary information using modern data storage technologies, saving in memory key-value pairs, and the project aims to select the best tools to support this

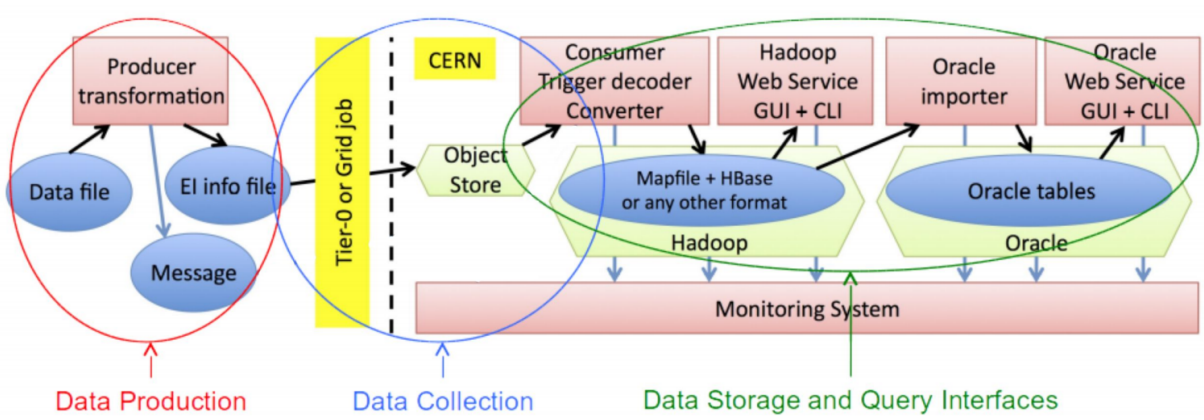

Figure 1. EventIndex architecture. The parts of the Data Production system are delimited with a red circle, a blue circle is used to mark the Data Collection system, and the Data Storage and query interfaces lie within a green circle.

application in terms of performance, robustness and ease of use. The resulting solution allows fast and efficient selection of events of interest, based on various criteria, from the billions of events recorded; and it also provides an indexing system that points to those events in millions of files scattered in a world-wide distributed computing system.

The system is designed to be a complete catalogue of ATLAS events, real and simulated data. Each event record contains three blocks of information.

- Event identifiers:
  - run and event number
  - trigger stream
  - luminosity block
  - bunch crossing ID (BCID)
- Trigger decisions:
  - trigger masks for each trigger level
  - decoded trigger chains (trigger condition passed)
- References to every event at each processing stage:
  - these are unique pointers to that event on the grid, enabling *event picking* jobs to retrieve specific events of interest

The infrastructure is currently being used for event picking (selecting single events that satisfy a set of conditions), data production and consistency checks, trigger checks (event counting based on trigger selection, trigger overlap detection), and event skimming.

The EventIndex architecture is composed of a data production system, a data collection system, a data storage system and a monitoring system, as it is shown in Figure 1. The Data Production system extracts EventIndex information from ATLAS files produced at CERN or on the Grid [4]. The process starts automatically as soon as a new dataset is registered in the ATLAS Metadata Interface database (AMI, [5]). These indexing jobs also provide a data integrity test, as they are the first to run on new data files. All new datasets containing events in AOD format (Analysis Object Data, i.e. the reconstruction output) are indexed by default, while other derived AOD formats (DAODs, i.e. selected events with reduced information for specific analyses) are only indexed on demand. Since the extracted metadata contains also the

references to the corresponding raw data, it is always possible to extract events in RAW data format too. For simulated data, all event generator files are indexed as well.

The extracted event metadata is packed into small files and transferred to the CERN Object Store by the Data Collection system [6, 7], which runs in servers at CERN. The Data Collection Supervisor system controls the transfers and validates indexed data. When a new dataset has been indexed, the Supervisor informs the Consumer processes, which then fetch data from the Object Store and store them in the Hadoop [8] cluster at CERN.

Data Storage units provide permanent storage for EventIndex data, fast access for the most common queries, and finite-time response for complex queries. The full information is stored in Hadoop in compressed MapFile format [9]. An internal catalogue in HBase [10] keeps track of the status of each dataset and holds dataset metadata. All event records are also inserted into a HBase table, which is used for fast event lookup for the event picking use case.

Real data records, containing only event identification information and pointers to event locations, are also copied to an Oracle database. The connection of this database with other ATLAS databases stored in the same Oracle cluster is especially useful to check dataset processing completeness and detect duplicates, providing added value to this information. Oracle is also much faster in event lookup operations, as long as the data schema has been well designed [11].

A monitoring system [12], implemented using InfluxDB [13] to store the data and Grafana [14] to display them, keeps track of the health of the servers and of the data flow, providing a visualisation of the system status and of the amount of stored data.

2. Evolution towards LHC Run 3

The current design of EventIndex, based on Hadoop and implemented using MapFiles and HBase, has been in continuous operation since 2015 with satisfactory results. Nevertheless, the current implementation of the EventIndex started showing scalability issues due to new use cases and the increasing amount of stored data. When the EventIndex was designed for Run 2, the RAW trigger rate for physics events was 200 Hz, increasing to 1 kHz by the end of Run 2, 7 kHz are expected in Run 3, and 10 kHz in Run 4. The fast data querying for real events based on Oracle is no longer sufficient to cover all requests. Therefore, an R&D programme was started to explore new technologies that would allow the EventIndex to include new functionalities and to keep up with the demanding event rates of the future.

A new design based on Apache Phoenix [15] queries and new HBase event table prototypes have been tested and shows encouraging results. Phoenix combines the power of standard SQL with the flexibility of the NoSQL world by taking SQL queries and compiling them into a series of HBase scans. However, in this approach the RowKey design of HBase tables needs to be adapted to Phoenix's types and sizes (losing some performance).

2.1. HBase optimisation and initial service provisioning

An extensive optimisation effort has been put forth in order to understand and optimise the configuration of HBase, leading to an improved performance in HBase 2.X installations.

The test cluster at CERN consists of 12 machines (16 physical cores, 64 GB of RAM) with CDH 5.15 [16] installed. The two tables considered had the following design:

- Datasets
  - RowKey design based on *dspid* and *dataType*, where *dspid* is an identifier based on $< project, runNumber, streamName, prodStep, version >$
- Events (salting enabled)
  - RowKey design based on *dspid* and *eventnumber*
  - Data block encoding: *Fast_diff*

  - Block compressor: Snappy
  - Salt buckets: 10

Apache Spark [17] was used to read from files and write to HBase via the Phoenix API.

A total of 103 billion event records were loaded from 50000 datasets, reaching an average loading speed of 80 kHz. The resulting size in HBase was 36.8 TB with an average record size of 392 B. The average data extraction time was measured to be 0.3 s.

During Phoenix testing, the first query on the event table was seen to take some time (13 s), while any other query afterwards would return within 1 s. Some of the tests with joint queries took too long and failed. This happens because joint queries can include full scans on composed data and can require a huge amount of memory. Phoenix developers are working to optimize the behaviour of joins. Nevertheless, it is safer and typically faster to make queries on datasets and events separately instead of joining them together.

2.2. Investigation of key composition

Phoenix and HBase work well when your application does point lookups and small range scans. This can be achieved by good primary key design

A good design of the primary key is paramount in order to improve the performance of the Phoenix-HBase pair, since these solutions are known to work better with small range scans. With this goal in mind, the EventIndex team proposes a binary composite key with the minimum number of components:

$$< (salt).dspid.dstypeid.dssubtypeid.eventno.seq >:< (1).4.1.1.8.2 > B$$

$$(Byte).Integer.TinyInt.TinyInt.BigInt.SmallInt$$

where,

- *salt* is the slicing function made internally by Phoenix,
- *dstypeid* and *dssubtypeid* are identifiers of the data type,
- *seq* allows insertion of duplicate events.

This key allows direct event peeking, range scanning over all the events for a given dataset, and range scan over all the events for all the streams with the same *dspid*. As a useful side effect, it defines a *canonical container*, which includes all the events that share the same *dspid.dstypeid*. The RowKey space is divided (*salt*) in order to improve write speed during the insertion of events for a dataset, so that all the writes dont go to the same region. The reverse bit order function is used to distribute keys evenly over the whole key space. All Int fields in the RowKey are unsigned.

HBase families are groups of columns, which represent related data that are stored together on the file system and should have the same general access pattern. Therefore, the performance of the following tests was evaluated separately for each of the families shown in Table 1.

Storage size after HBase compression was tested on a subset of the data available (224,389,536 event records). Snappy [18] compression reduced the final size by a factor 4. Additionally, reducing the column names to 1 letter was found to improve the used space, but not enough to justify the loss of human readability. Therefore, it was decided to reduce names only to a meaningful minimum.

Several alternatives for storing trigger information were tested:

- Binary packing in two codings:
  - using 10 (12) bits for L1 (HLT). (TC)
  - using 10 (12) bits for L1 (HLT), and packing the components together.
- Native Phoenix arrays (smallInt []):

Table 1. Event table schema.

Family	Name	Type	Definition
A: event location	guid	binary(16)	GUID of file
A: event location	oid1	unsigned_int	
A: event location	oid2	unsigned_int	
B: event provenance	prov	varchar(10000)	Provenance (JSON)
C: event description	smk	unsigned_int	Trigger SuperMasterKey
C: event description	hltpsk	unsigned_int	HLT prescaler key
C: event description	l1psk	unsigned_int	L1 prescaler key
C: event description	lumiblocknr	unsigned_int	
C: event description	bunchid	unsigned_int	
C: event description	eventtime	unsigned_timestamp	
C: event description	lvl1id	unsigned_long	
D: level 1 trigger (L1)	lumiblocknr	unsigned_int	
D: level 1 trigger (L1)	bunchid	unsigned_int	
D: level 1 trigger (L1)	tbp	unsigned_smallint[]	Trigger before prescaler
D: level 1 trigger (L1)	tap	unsigned_smallint[]	Trigger after prescaler
D: level 1 trigger (L1)	tav	unsigned_smallint[]	Trigger after veto
E: high level trigger (HLT)	lumiblocknr	unsigned_int	
E: high level trigger (HLT)	bunchid	unsigned_int	
E: high level trigger (HLT)	ph	unsigned_smallint[]	HLT Physics
E: high level trigger (HLT)	pt	unsigned_smallint[]	HLT Passtrought
E: high level trigger (HLT)	rs	unsigned_smallint[]	HLT Resurrected

  - using one array per trigger: TBP, TAP, TAV, PH, PT, RS
  - using just one array per trigger level (L1 and HLT)
    - storing all components together with their sizes at the end.
- Using also TPB*, TAP*, TAV*, which eliminates triggers present in next stages. In L1 trigger, TAP and TAV stages are subsets of the previous stage, i.e. $TBP \supseteq TAP \supseteq TAV$. Space can be saved if instead of storing the whole stage, only the difference with respect to the next stage is stored.

All the data block encoding types supported by HBase were considered in the tests: *Prefix, Diff, Fast_diff and Prefix_tree*. The resulting size in the Hadoop file system (HDFS) was tested using 10 million events. HDFS file sizes were read after compaction. File redundancy was not taken into account in these tests.

The tests showed that packing the trigger components together into binary data is the best method. Using one array to store all triggers for a level was slightly worse than using one array per trigger, but this approach was finally chosen because it uses standard Phoenix coding.

Storing information while eliminating the triggers present in next stages reduced the file size.

Fast_diff was found to be better for *event location* and *event provenance*, while *Diff* was slightly better for *event description*, *L1* and *HLT*. Since the difference was small (1-8%), *Fast_diff* was the encoding chosen.

3. Conclusion

The EventIndex project started in 2012 at the end of LHC Run 1 driven by the need of having a functional event picking system for ATLAS data. The data storage and search technology selected in the first phase of the project (Hadoop MapFiles and HBase) was the most advanced available at that time in the fast-growing field of BigData and indeed proved reliable and performed satisfactorily. Nevertheless, the current implementation of the EventIndex started showing scalability issues as the amount of stored data increased.

Phoenix queries and new HBase event table prototypes have been tested. The table schema selected for the final prototype contains a binary composite RowKey with the minimum number of components, and column names' size reduced to a meaningful minimum. All trigger components are packed together into binary data and one array is used to store all triggers for a level. Triggers present in next stages are eliminated before the information is stored. HBase Snappy compression and the *Fast_diff* block encoding are used to further reduce storage size. The new prototype with basic functionality is ready and shows encouraging results. Effort is now focused on optimising the performance of the prototype and on improving the design of the interfaces.

The plan is for the new system to be operational during 2019 in parallel with the old one, and to phase out the old system during 2020 (well in advance of the start of LHC Run 3).

References

[1] ATLAS Collaboration (2008) The ATLAS Experiment at the CERN Large Hadron Collider, JINST 3 (2008) S08003
[2] Barberis D et al. (2014) The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of data, J. Phys.: Conf. Ser. 513 042002
[3] Barberis D. et al. (2015) The ATLAS EventIndex: architecture, design choices, deployment and first operation experience, J. Phys. Conf. Ser. 664 042003
[4] Bird I. et al. (2005) LHC Computing Grid Technical Design Report, CERN-LHCC-2005-024, http://lcg.web.cern.ch/LCG/tdr
[5] Albrand S. (2010) The ATLAS metadata interface, J. Phys. Conf. Ser. 219 042030, doi:10.1088/1742-6596/219/4/042030 J Fulachier et al. (2017), ATLAS Metadata Interface (AMI), a generic metadata framework, J. Phys.: Conf. Ser. 898 062001
[6] Sánchez J et al. (2015) Distributed Data Collection for the ATLAS EventIndex, J. Phys.: Conf.Ser. 664 042046, doi:10.1088/1742-6596/664/4/042046
[7] Fernandez Casani A et al. (2018) Distributed Data Collection for the Next Generation ATLAS EventIndex Project, ATL-SOFT-PROC-2018-039
[8] K Shvachko et al. (2010) The Hadoop Distributed File System, DOI: 10.1109/MSST.2010.5496972, http://hadoop.apache.org
[9] Favareto A et al. (2016) Use of the Hadoop structured storage tools for the ATLAS EventIndex event catalogue, Phys. Part. Nuclei Lett. 13: 621, doi:10.1134/S1547477116050198
[10] HBase: https://hbase.apache.org
[11] Gallas E et al. (2017) An Oracle-based EventIndex for ATLAS, J. Phys.: Conf. Ser. 898 042033
[12] Alexandrov E et al. (2018) BigData tools for the monitoring of the ATLAS EventIndex, http://ceur-ws.org/Vol-2267/91-94-paper-15.pdf
[13] InfluxDB: https://www.influxdata.com
[14] Grafana: https://grafana.com
[15] Phoenix: https://phoenix.apache.org
[16] Cloudera: https://www.cloudera.com
[17] Spark: https://spark.apache.org
[18] Google Snappy: https://github.com/google/snappy