The ATLAS EventIndex: architecture, design choices, deployment and first operation experience

D. Barberis*, S.E. Cárdenas Zárate, J. Cranshaw, A. Favareto, Á. Fernández Casaní, E.J. Gallas, C. Glasman, S. González de la Hoz, J. Hrivnáč, D. Malon, F. Prokoshin, J. Salt Cairols, J. Sánchez, R. Többicke* and R. Yuan
on behalf of the ATLAS Collaboration

1 Università di Genova and INFN, Genova, Italy
2 Universidad Técnica Federico Santa María, Valparaíso, Chile
3 Argonne National Laboratory, Argonne, IL, United States
4 Instituto de Física Corpuscular (IFIC), Univ. de Valencia and CSIC, Valencia, Spain
5 University of Oxford, Oxford, UK
6 Universidad Autónoma de Madrid, Madrid, Spain
7 LAL, Université Paris-Sud and CNRS/IN2P3, Orsay, France
8 CERN, Geneva, Switzerland

*Corresponding author: Dario.Barberis@cern.ch

Abstract. The EventIndex is the complete catalogue of all ATLAS events, keeping the references to all files that contain a given event in any processing stage. It replaces the TAG database, which had been in use during LHC Run 1. For each event it contains its identifiers, the trigger pattern and the GUIDs of the files containing it. Major use cases are event picking, feeding the Event Service used on some production sites, and technical checks of the completion and consistency of processing campaigns. The system design is highly modular so that its components (data collection system, storage system based on Hadoop, query web service and interfaces to other ATLAS systems) could be developed separately and in parallel during LS1. The EventIndex is in operation for the start of LHC Run 2. This paper describes the high-level system architecture, the technical design choices and the deployment process and issues. The performance of the data collection and storage systems, as well as the query services, are also reported.

1. Introduction
During LHC Run 1 (2009–2013), the ATLAS Collaboration [1] used a global event catalogue (called “TAG Database”), implemented in an Oracle database with an associated web service as front-end for the users. The TAG DB fulfilled its primary objective to satisfy the “event picking” use case and was also useful for checking the completeness and consistency of data processing cycles, thus identifying production problems that could lead to missing or duplicate events. One limitation of the TAG DB was the lack of flexibility of the database schema, which was set prior to the start of LHC data-taking and was hard to modify afterwards to follow the needs of the collaboration. Another limitation was that the system could not easily add information about processing stages downstream of AOD production (which is the data format from which analysis datasets are derived).
In late 2012, ATLAS launched a prototyping project to design a new system to catalogue all real and simulated data, in all processing stages [2]. The new system has to scale to the order of several $10^{10}$ events (the number of events expected for Run 2 between 2015 and 2018), be flexible in its schemas to accommodate a variety of quantities to be stored that could change in the future, use established and possibly open-source technologies and be “easy” to develop, deploy and operate.

The use cases that were identified and analysed in the project design phase were:

- Event picking: given a list of run numbers and event numbers, trigger stream, event format and processing version\(^1\), find the events and return pointers to them to the user that issued the query, who can then use the data management tools to retrieve them.
- Event Service: this is a new component of the ATLAS Distributed Computing infrastructure that can distribute single events to processors of any type that run ATLAS production jobs [3]. It needs for each input file the internal references (pointers) of each contained event in order to extract events one-by-one and pass them to the processing units.
- Trigger checks and event skimming: the population of events that passed given triggers and of events that passed multiple triggers can be retrieved from the event catalogue. Similarly a trigger-based event selection can be done, retrieving the references to the selected events and then the events themselves.
- Production consistency checks: each production cycle should be checked for completeness (the number of produced events is the same as the number of input events) and consistency (no duplicate events).

The studies performed in 2012-2013 resulted in the high-level design of the new ATLAS EventIndex catalogue [4]. This paper reports on the system architecture and the design and implementation choices for each component. The initial deployment and operation experience, as well as preliminary performance evaluations, are also discussed.

2. System architecture

In order to bring the new system into production operation as quickly as possible, it was decided to store only the information for each event that is actually needed to satisfy the above use cases. The project was divided into work packages that could be developed almost independently from each other (after having defined the interfaces), and choosing simple and robust technological solutions for each component.

The minimal information to be stored for each event consists of:

- Event identification: run number, event number, trigger stream, event format and processing version. For real data, the “luminosity block”\(^2\) number is also stored as it can be used to link to detector or trigger condition information. For simulated data, the “simulation process number” is added to be able to uniquely identify each event.
- Trigger information: the list of trigger chains passed by the given event.
- References of the event: the GUIDs (Global Unique IDentifiers) of the logical files that contain the given event, plus the internal pointers within these files. Users can use the GUID to find the physical files containing the event by querying the ATLAS distributed data management system Rucio [5], and the internal pointers to get the actual event and process it in whichever way they need.

The work packages provide the functionality needed to operate each stage of the EventIndex data flow:

---

\(^1\) The combination of run number, event number, trigger stream, event format and processing version uniquely identifies an ATLAS event.

\(^2\) A luminosity block is a short unit of time, during Run 1 normally one minute, during which all data-taking conditions and calibrations are assumed to be constant.
• The Data Collection system collects data from jobs at Tier-0 or on the Grid that produce new data. The EventIndex information for each permanent output file is transmitted to a central server at CERN where it is validated, reformatted and stored in the EventIndex storage system.
• The Core Storage system accepts data from the Data Collection system, optimizes the data format in order to be efficient in satisfying the most common queries, creates the internal catalogue and indices as needed, physically stores the data in its own storage space, accepts queries from the front-end web server and returns the results.
• The Trigger Decoding service unpacks the trigger information of each event and makes it readily available in the event records.
• The Query Server is a web service that acts as a front-end to the storage system. It provides a command line interface and a web interface that can be used to find and retrieve the stored information.
• The monitoring system provides continuous information on the health and load of all the servers involved, as well as on the data traffic and query response times.

Figure 1 shows a sketch of the data flow in terms of ATLAS and EventIndex components.

Figure 1: Block diagram of the EventIndex data flow. Information is produced by jobs running on the Tier-0 or on the Grid, inserted and catalogued into the database, and retrieved by interactive and batch processes.

Each component is developed in the language that is best suited for the task and the environment of that component: Python is used for the Data Collection system and Java for the code that is connected to the core storage service.

3. Data collection
The task of the Data Collection system is to extract the information on the events to be catalogued from each file and pass it to the central catalogue. There are three components of this system: the Producer process, the messaging system used to transmit the information, and the Consumer process.

The EventIndex Producers can run in the Tier-0 cluster or on any ATLAS Grid site, as stand-alone jobs or as part of more complex workflows. Stand-alone jobs read and extract information from existing files that were previously produced, for example all existing Run 1 data; for the data files that are currently produced, it is more convenient to run the Producer as part of the same job that creates the event files, as the last job step. In both cases the Producer, which is a Python script that uses some C++ code from the ATLAS Athena framework to extract event information, reads the headers of each event in the file and saves the relevant information in a local SQLite3 [6] file. At the end of the job, the SQLite3 file is read back, serialized, compressed and encoded in JSON format, split into separate units of 10 kB size, and sent to the messaging server at CERN using the ActiveMQ [7] messaging service with the STOMP [8] protocol.

ActiveMQ was chosen as the data transport system as it is supported by the CERN IT department and its performance and robustness has been measured to be sufficient for the expected data flow.

The EventIndex Consumer processes run in dedicated servers at CERN. They read the messages from the ActiveMQ broker, recombine them to reconstruct the information relative to each data file and save this information in a temporary storage space. A validation process checks that all messages
have been received correctly; when all EventIndex files for all jobs of a given task have been received and validated, and the task has been completed successfully, the EventIndex information is formatted for permanent storage.

Figure 2 shows a sketch of the data flow and at the same time of the building blocks of the Data Collection system. More details on the Data Collection components are available in Ref. [9].

4. Core storage system and query services

The experience with the TAG DB during Run 1, which used Oracle as back-end storage, showed that a more flexible storage system had to be selected for the EventIndex. A study by the WLCG Collaboration in 2012 [10] showed that several structured storage systems (improperly called “NoSQL databases”) could be satisfactorily used for applications like the EventIndex, where large numbers of small data records are stored and have to be searched and retrieved efficiently. Out of these, the Hadoop [11] ecosystem provides a number of data storage options and auxiliary tools that can be used to optimize the data format in order to make the most common query types very efficient. As CERN decided at the same time to support Hadoop at the system level and provide a cluster for large applications like the EventIndex, Hadoop was naturally selected as the baseline back-end storage system.

A few data formats were tested early on, from simple CSV files in Hadoop to the extreme opposite of dumping all data into an HBase [12] table. The selected format has the data in Hadoop “Mapfile” format (“TagFiles”). The Mapfile format is a special Hadoop storage format where the keys are stored in an index file and kept in memory for fast access, and values are stored in standard sequential data files on disk. The keys are ordered so that key-based search operations are immediate. Some TagFiles contain full EventIndex records, others contain only references to other data; this is transparent to users, as all files are treated in the same way. A “TagSet” is a set of TagFiles. It can be treated in the same way as a single TagFile. Search results are stored (as TagFiles) to be available for later reuse.

Three ways of searching are possible:

- **Key-based search on (sequence of intervals of) keys:** it gives almost immediate results. The primary keys consist of RunNumber-EventNumber pairs. Further selections can follow a key-based search.
- **Full MapReduce search:** it has a minimum of 1-minute initialization time to set up the MapReduce jobs but all stored data are available to the search. The search time depends on the amount of data read in by the MapReduce jobs and (most importantly) by the amount of data written out to the resulting data file.
- **Full-scan search:** it is the slowest, but is useful in the development phase to understand what is going on during the search.
Users can request an accumulation of statistics (for example the number of entries satisfying the search conditions) instead of the delivery of the full result. All results are stored within the system and can be retrieved later. Users can also specify the required output format and contents. As the Hadoop framework is written in Java, the search and formatting options can contain any legal Java code using TagFile variables.

When data are imported from the temporary storage space where they had been put by the Consumer, the TagConverter process creates the TagFiles; one TagFile corresponds to one data file that has been indexed. Files are organized in directories whose names are derived from the dataset name of the indexed file. In this way, the dataset information (or part of it) can be used to reduce the search range to the minimum, making the search more efficient. On the other hand, Hadoop is much more efficient at storing and reading large files rather than many small files; it is therefore convenient to have additional representations of the data, namely a single TagFile for all events in a dataset, and a single TagFile for all events in a “campaign”\textsuperscript{3}. In this way, queries can be directed by the front-end server to the data representation that is most suitable according to the query characteristics. These additional representations need of course more storage space but as long as it remains in the limits of 10s of TB the trade-off between additional disk space and performance is largely positive.

An additional data representation is provided to minimize the response time for the queries originated by the Event Service, which retrieves the run numbers, event numbers, GUIDs and internal references for all events in a given data file or dataset. In this way, data are pre-sorted so that the retrieval of the required information from the catalogue is immediate.

More details on the Core Storage and Query Services components are available in Ref. [13].

5. Trigger information decoding

The ATLAS trigger system had three selection levels during Run 1: Level-1 (implemented in specialized hardware), Level-2 (software analyzing only information within the “regions of interest” marked by Level-1) and the Event Filter (software analyzing the complete event). For Run 2 the software triggers have been merged into a single unit: the High-Level Trigger. The combination of a few trigger levels, many trigger signatures (detection of single particles or global event quantities), and the possibility to apply pre-scale factors to each signature, result in a large number, of order 1000, of possible trigger chains that can be satisfied by any recorded event. In addition the list of active triggers and their pre-scale factors is defined at the start of each run but can vary during the run according to the LHC luminosity and detector conditions. In reality the number of satisfied trigger chains for a single event is a few tens, as some are subsets of others (for example with higher thresholds) and some are highly pre-scaled monitoring triggers.

Each event record contains the reference (“Super Master Key” or SMK) to the list of configured triggers for its run and the trigger chain numbers of the satisfied triggers. The correspondence between the trigger chain numbers and their names and parameters, as functions of the SMK, is kept in the Trigger Database; this correspondence is the same for all events of the current run (and all runs with the same SMK). The Trigger DB is therefore needed to decode the trigger information of each event and be able to use it for search and count operations.

The COMA (COnditions MetadatA) [14] database in Oracle contains all relevant metadata needed by the EventIndex from the Trigger DB. In order to have all information in the same environment and to minimize the calls from Hadoop to Oracle, the trigger information is copied (and periodically updated) from the COMA database in Oracle to an HBase table in Hadoop. The TagConverter process that creates the TagFiles in Hadoop can thus use this information directly from HBase to decode the trigger pattern of each event and store in the TagFiles the list of trigger chain names that were satisfied by this event. Trigger chain names (character strings) can then be used as search terms for queries.

\textsuperscript{3} A “campaign” in ATLAS data processing is the union of all datasets with the same format that contain events recorded with the same data conditions and processed using the same software version; for example all proton-proton collisions taken in 2012 at a centre-of-mass energy of 8 TeV and processed in real time at the Tier-0.
Figure 3 shows a sketch of the trigger data flow and decoding system. More details on the trigger decoding data flow are available in Ref. [15].

In principle it is possible to index the data by trigger chain, in order to speed up all searches involving them. This need has not arisen yet but all tools are available in case this use case will become more important or such searches will be more frequent than now foreseen.

6. System monitoring

Any software system needs a comprehensive suite of monitoring tools that provide information on automatic operations and can send alarms in case of recognized problems. For the EventIndex there are three areas of monitoring:

- Monitoring the status and activities (CPU load, memory usage, I/O rates, local disk occupancy) of all servers in the chain. This includes the Hadoop and ActiveMQ servers managed by CERN-IT and the virtual machines that host EventIndex services (Consumer, TagConverter, web server for user access).
- Monitoring the operations performed by EventIndex processes, including the input data flow, occupancy of the buffers, query rates and response times, total disk space used in Hadoop, Trigger Database statistics and consistency information.
- Consistency checks of the production and EventIndex data, comparing the numbers of events in input and output files for processing steps to the EventIndex records.

Low-level monitoring tools were initially implemented in the CERN Service Level Status (SLS) [16] framework and subsequently ported to the new CERN framework [17] based on Kibana [18]. Operation monitoring and consistency checks were implemented directly in the new framework. More details on EventIndex system monitoring are available in Ref. [15].

7. Deployment and performance

After almost two years of development and intensive testing, the deployment phase started in autumn 2014. The test systems were progressively made more robust and turned into a production service. Currently (spring 2015) the following services are used:

- The ActiveMQ service provided by CERN-IT, consisting of three brokers (two in Geneva and one in the CERN Computing Centre at Wigner Institute, Budapest, for redundancy and robustness in case of network problems).
- The Hadoop service provided by CERN-IT, consisting of 16 servers (including the head node) for the time being, but soon to be upgraded.

Figure 3: Trigger information data flow and decoding system.
• Two virtual machines running the Consumer processes.
• Two virtual machines running the TagConverter processes and the Query Services (the latter being a web server).

Filling the EventIndex started in January 2015, with Run 1 data. It was decided to index first all Tier-0 productions (which give also the references to RAW data) and then the last version of reprocessed data, discarding all intermediate reconstruction versions, as they are no longer of interest. This operation needed the recall of large amounts of data from tape at CERN and all Tier-1 centres, which determined the filling rate and at the same time prevented stressing any of the hardware or software components of the EventIndex. The Tier-0 data collection was completed in April 2015, whereas the data collection from reprocessed data at Tier-1s started on the Grid but, having lower priority than other ATLAS computing activities, is still in progress (April 2015). Run 1 data consist of about 6 billion events, which correspond to approximately 2 TB of EventIndex data (single version, before replication). Figure 4 shows the data collection statistics for Tier-0 Run 1 processed data.

![Figure 4: Filling the EventIndex with Run 1 Tier-0 processed data during February-March 2015.](image)

Cosmic ray data taken in 2014-2015 were indexed at Tier-0 in preparation for the start of Run 2. The EventIndex Producer is run so far still in stand-alone mode, i.e. as an independent job that reads AODs produced at Tier-0 and records also the references to each event in RAW and ESD format. Work is in progress to enable running the Producer as the last step of any job that produces permanent files that will be recorded in the data management system Rucio [5], excluding the temporary files that are generated within a given workflow or will be merged into larger files at the end of the given process.

Tests of the Event Service [3] for simulated events need data in EVNT (event generator output) format to feed detector simulation processes that run on high-performance computers (supercomputers) or other external resources. These events are so far indexed on request but the plan is to automate this operation mode soon.

An additional request by ATLAS is to index all outputs of the new Derivation Framework [19] and create a matrix counting the overlap of events that are selected by multiple derivation algorithms. Data need to be collected by Grid jobs and then sorted; for each event the EventIndex can record this “offline trigger” and then a specialized algorithm can fill a matrix containing in each cell the number of events having satisfied a given pair of offline triggers. Once done for Run 1 data, these results will be used to optimize the derivation algorithms for Run 2.

As expected, the response time of different queries differs substantially for queries that are satisfied by the internal catalogue and those that need to launch a MapReduce task, and the measured times are approximately proportional to the amount of data to be searched (read in from disk) and the amount of data to be retrieved (written to the output file). Taking as reference a dataset with 123.5 million events, simple RunNumber-EventNumber searches need 30 seconds, counting all events almost 5 minutes, and retrieving all information over one hour (but this is not a “normal” user query).
8. Conclusions and outlook
The first ideas leading to the EventIndex project were discussed in ATLAS in Autumn 2012. Two-and-a-half years later, the EventIndex exists and the deployment phase, including back-filling it with all Run 1 data, is almost completed. All fundamental building blocks perform as expected. Work is in progress to turn deployment into completely automated operations, including the completion of the integration of the Producer with data processing operations, more automatic data validation, increased robustness against network problems and hardware failures, additional internal monitoring, and performance (timing) improvements for common queries. This work is expected to be completed during 2015; after that time the monitoring tools will be robust enough to be used by general computing shifters and the active work by experts will be reduced to occasional advice and interventions in case of problems.

Acknowledgements
The authors wish to acknowledge their funding agencies for support for this work: INFN (Italy), CONICYT (Chile), DOE (USA), MINECO (Spain), STFC (UK), CNRS/IN2P3 (France) and CERN. This research is partially financed by MIUR (Italy) through project PRIN STOA-LHC 20108T4XTM and by the ePlanet Project (EU).

References
[1] ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider, *JINST* 3 S08003 doi:10.1088/1748-0221/3/08/S08003
[2] Barberis D et al. 2014 The future of event-level information repositories, indexing, and selection in ATLAS, *J. Phys. Conf. Ser.* 513 042009 doi:10.1088/1742-6596/513/4/042009
[3] Wenaus T et al. 2015 The ATLAS Event Service: a new approach to event processing, *J. Phys. Conf. Ser.*, these proceedings.
[4] Barberis D et al. 2014 The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of data, *J. Phys. Conf. Ser.* 513 042002 doi:10.1088/1742-6596/513/4/042002
[5] Garonne V et al. 2014 Rucio – The next generation of large scale distributed system for ATLAS Data Management, *J. Phys. Conf. Ser.* 513 042021 doi:10.1088/1742-6596/513/4/042021
[6] SQLite3: https://www.sqlite.org
[7] ActiveMQ: http://activemq.apache.org
[8] STOMP: http://stomp.github.io
[9] Sanchez J et al. 2015 Distributed Data Collection for the ATLAS EventIndex, *J. Phys. Conf. Ser.*, these proceedings.
[10] Barberis D et al. 2012 Report of the WLCG Database Technical Evolution Group, CERN Report
[11] Hadoop: http://hadoop.apache.org
[12] HBase: http://hbase.apache.org
[13] Hřívňáč J et al. 2015 QuerySpaces on Hadoop for the ATLAS EventIndex, *J. Phys. Conf. Ser.*, these proceedings.
[14] E J Gallas et al. 2012 Conditions and configuration metadata for the ATLAS experiment, *J. Phys. Conf. Ser.*, 396 052033 doi:10.1088/1742-6596/396/5/052033
[15] Prokoshin F et al. 2015 Integration of the EventIndex with other ATLAS systems, *J. Phys. Conf. Ser.*, these proceedings.
[16] Lopienski S 2008 Service level status—a new real-time status display for IT services, *J. Phys. Conf. Ser.* 119 052025 doi:10.1088/1742-6596/119/5/052025
[17] CERN monitoring: https://cern.service-now.com/service-portal/sls.do
[18] Kibana: https://www.elastic.co/products/kibana
[19] Catmore J et al. 2015 A New Petabyte-scale Data Derivation Framework for ATLAS, *J. Phys. Conf. Ser.*, these proceedings.