Available online at www.sciencedirect.com
ScienceDirect
Nuclear Physics B Proceedings Supplement 00 (2014) 1–6
Nuclear Physics B Proceedings Supplement

The ATLAS EventIndex: Full chain deployment and first operation

D. Barberis[a], J. Cranshaw[b], A. Favareto[a], A. Fernández Casaní[c,1,*], E. Gallas[d], S. González de la Hoz[c], J. Hřivnáč[e], D. Malon[b], M. Nowak[f], F. Prokoshin[g], J. Salt[c], J. Sánchez Martínez[c], R. Többicke[h], R. Yuan[e]

[a] *Università di Genova and INFN, Genova, Italy*
[b] *Argonne National Laboratory, Argonne, IL, United States*
[c] *Instituto de Física Corpuscular (IFIC), Valencia, Spain*
[d] *Oxford University, Oxford, United Kingdom*
[e] *Université Paris-Sud and CNRS/IN2P3, Orsay, France*
[f] *Brookhaven National Laboratory, Upton, NY, United States*
[g] *Universidad Técnica Federico Santa María, Valparaíso, Chile*
[h] *CERN, Geneva, Switzerland*

Abstract

The Event Index project consists in the development and deployment of a complete catalogue of events for experiments with large amounts of data, such as the ATLAS experiment at the LHC accelerator at CERN. Data to be stored in the EventIndex are produced by all production jobs that run at CERN or the GRID; for every permanent output file, a snippet of information, containing the file unique identifier and the relevant attributes for each event, is sent to the central catalogue. The estimated insertion rate during the LHC Run 2 is about 80 Hz of file records containing ~15 kHz of event records. This contribution describes the system design, the initial performance tests of the full data collection and cataloguing chain, and the project evolution towards the full deployment and operation by the end of 2014.

Keywords: ATLAS, Distributed Computing
PACS: 29.50.+v, 29.85.Fj

1. Introduction and objectives

Experiments like ATLAS [1][2] produce large amounts of data that need cataloguing for a later access by the physicists. During 2011 and 2012 ATLAS produced 2 billion real events and 4 billion simulated events per year. In addition, events were reprocessed and stored in several file formats, increasing numbers to 138 billion (10^9) real event instances, and 93 billion simulated events in year 2012 alone. Therefore in order to recall selected events from data storage systems, it is necessary to have a system that contains the reference to the file including every event at every stage of processing. ATLAS already has an event database (TAGDB) implemented in Oracle, as that was the only proven technology that could hold the impressive amount of expected data when this project started, long before the beginning of LHC operations [3][4][5][6]. The main purposes of this database were the selection of events on the basis of physical variables, the identification of files containing a list of events already selected by other means, and consistency checks on the completeness of event processing. In the TAGDB, each event is recorded several times, once for each reconstruction cycle, and contains three main data blocks: information identifying the event (event and run number and trigger decisions), information allowing the localization of the event data in files that contain it, and physical variables of the event, such as the number of reconstructed particles, their identification and properties, which might be useful to select events for specific analyses. The

*Corresponding author
[1] On behalf of the ATLAS Collaboration.

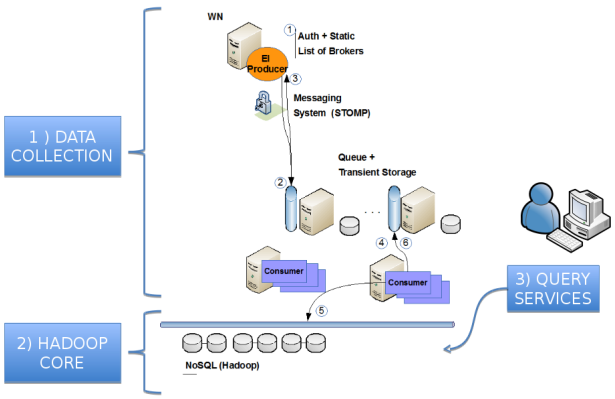

Figure 1: EventIndex prototype.

TAGDB, with all related services and user interfaces, works for the retrieval of the event information for limited numbers of events and for the technical checks described above; the event selection use case by physics variables turned out not to be generally useful because sometimes those quantities changed after the initial reconstruction and there was no mechanism in the system to update them. Furthermore, the TAGDB is slow, intrinsically complex and at times unreliable. In addition, the enormous amount of data (1 kB per record, which turns into several tens of TB in Oracle for all data collected by ATLAS so far) makes the implementation in Oracle particularly labour-intensive and expensive [7]. Oracle storage capacity was deployed to store the ˜8 billion real and simulated events recorded during the LHC Run 1; future data taking rates will be higher, exceeding the limits of Oracle scalability required by the existing workflow [8][9][10]. The TAGDB is potentially very useful but has to improve. In order to overcome the previous limitations, we tried to follow a modern approach like using structured storage, and in particular NoSQL technologies [11]. These technologies are more appropriate to scale for increased data rates like we will see at LHC in Run 2 starting in 2015. Moreover, they are designed for usage with commodity hardware, and also with fixed cost per unit, and great scalability. They are cheaper, easier to use and faster for sparse searches over large amount of data, so we are using them for the future event cataloguing system, within the EventIndex project.

2. EventIndex project

With the EventIndex project we want to overcome the previous limitations and to use NoSQL technologies to provide a complete catalogue of all ATLAS events, real and simulated data, and for all the stages. Each reconstruction campaign produces new versions of every event, in different formats. For ATLAS, they are ESD (Event Summary Data, a full reconstruction output), AOD (Analysis Object Data, a summary of reconstruction to be used for physics analyses) and several versions of ROOT n-tuples [12][13]. For each reconstruction campaign and for each format, one can generate a key-value pair (where the value is always the navigational information, i.e. the identifier of the file that contains the event in question and the internal structured pointer) and add it to the original record. The result is that an event will always correspond to one and only one logical record in the database containing the entire processing history of the event.

Basically the use cases that we want to solve are the following:

1. Event picking: give me the reference (pointer) to "this" event in "that" format for a given processing cycle.
2. Event skimming: Give me the list of events passing this selection and their references.
3. Production consistency checks: technical checks that processing cycles are complete (event counts match).
4. Panda event service [14]: give me the references (pointers) for the events on the file identified by this GUID [15], to be distributed individually to processes running on (for example) HPC or cloud clusters.

3. First prototype

Figure 1 shows the different parts of the first prototype. The Data Collection task deals with the collection

of the worldwide production of the EventIndex information, to be stored by the Hadoop [22] core services. It follows a Producer/Consumer architecture, with the producers running at Tier-0 (CERN) and Grid sites, and the consumers at CERN to store centrally the information. Because of the large amount of data being continuously processed, care must be taken in the development and optimization of these applications. The whole transmission chain of this information must be optimized in order to sustain rates of the order of 10 to 100 kHz. The system must also include cross-checks to assure catalogue completeness (no missing processing stages for any subset of events). These aspects of performance and scalability are extremely important to have a product that will be useful to the scientific community. The Hadoop core task provides the infrastructure to load and host the EventIndex data, and provides the interfaces for the Query Services to implement the required use cases. The Query Services provide the tools for final users to access the EventIndex data, a catalogue that can be searched with different criteria.

4. Data collection producer

The Producer is the piece of software that is in charge of producing the EventIndex information and send it with a messaging system to the central brokers. The EventIndex information is produced with python scripts running in the Athena framework [16], inside pilot jobs that are sent worldwide to Tier-0 and Grid sites. For every permanent output file that has to be indexed, a snippet of information containing the file unique identifier and the relevant attributes is produced. The data producer process is split in two parts: the first part reads the data file to create an intermediate EventIndex file; the second one reads the information from the EventIndex file, builds the messages and sends them to the broker. The EventIndex file is used as an intermediate storage to decouple the processing of the input file from sending the messages. In the current prototype, the first step consists of a Python script that reads the event file and extracts the relevant information to be transmitted (event identification, trigger masks, references to the files containing the event). These scripts include tools for analyzing POOL, RAW and TAG formats [9]. The EventIndex file is a SQLite3 [17] file of Python serialized objects, containing key-value pairs in one table "shelf". The second step is a Python script that reads the EventIndex file, builds a sequence of messages (about 10 kB each) and sends them to the broker. For this first prototype, we tested the production of the EventIndex information in the grid, submitting several production jobs with an automated tool like prun [18]. The original measurements from production jobs in May 2013, during the LHC shutdown, estimated a number of 256k grid jobs per day, summing up the production of 300 million events every day. The current EventIndex information stored is about 1 kB/event, which means around 300 GB per day to be stored. A rate of the producers is therefore measured as 20 Hz of file records containing around 3.4 kHz of event records, to be increased in the following years up to 80 Hz of files and 30 kHz of event records.

5. Data collection messaging

Producers send the produced event information with a messaging system based on the text-based STOMP protocol [19], using JSON [20] encoding in every message in a compact form. Messages are received by one or more ActiveMQ [21] brokers. During these first tests of the prototype we were using just one broker at CERN, capable of handling all the messages, but we have planned to have another broker at the Wigner Institute site (in Budapest) to achieve high availability. The message size is set small, from 1 to 10 kB per message, to keep the broker queues agile and because messaging systems are usually designed for dealing with thousands or millions of small messages. Usually we have to divide the payload corresponding to an output file into several messages, and in this case, we use atomic transactions on the producer to logically group them. This allows processing all of them, so if there is any error or connection break, no partial processing occurs. In this latter scenario, a resubmission of all the messages is performed. The same broker can receive messages from several producers at the same time, so in the queue they may be mixed, but this will be later solved by the consumers as we will see. During our benchmarks and tests, the performance was enough to deal with the proposed rates, reaching 350 messages per second and a throughput of 10 MB/s per producer. When sending real events we reached 200k events/s and 60 MB/s (this test scenario used 1 broker, 6 producers and 4 consumers, with 50k events per job).

6. Data collection consumer

Consumers are in charge of retrieving the messages from the brokers, process them, and insert the EventIndex information into the Hadoop [22] testbed that is also used by the core service. These processes are run in one machine in the same network domain as the Hadoop

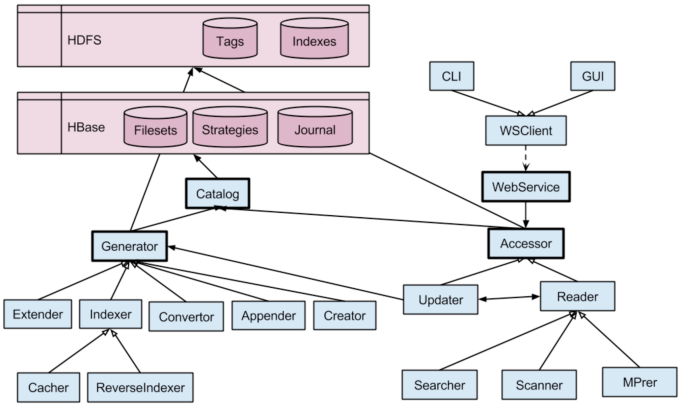

Figure 2: Hadoop core system.

testbed, so access is easier and also performance is better. Depending of the necessities the number of consumers can be increased, and run in several machines, but for our current tests one machine is enough to process all the messages. We have seen that messages can be originated by different producers and mixed in the queue, which means they will also be mixed when received by the consumers, and also potentially different consumers could receive messages from one producer. In order to solve this problem and process messages coherently, we use a messaging facility which is group tagging. This ensures that all messages from one particular producer will go to one particular consumer, while the latter is alive. For every message received, the consumer decodes the JSON [20] format and orders the events in memory. When all the events from an original output file are received, or when a threshold is reached, the events are written into Hadoop HDFS [23] in Mapfile format [24].

7. Hadoop core architecture

The Hadoop core services for the EventIndex are developed on top of the Hadoop infrastructure provided by the CERN-IT Hadoop service. Data are continuouly received and written in Mapfiles that may be later upgraded into some of their sub-formats in case further optimisation is needed. The event attributes are stored in several groups according to their nature and access pattern (constants, variables, references, physics). The data can be indexed in various ways (for example, using inverted indices for the trigger information). Some index files will be created at upload, others will be added later. Index files will just have a key plus references to data files. The searches can be then performed in two steps:

1. get reference from the index file;
2. using that reference, get the data.

Searches can be performed using keys, with immediate results (as keys are in memory), or with full searches on file-based data. Access to the data is achieved by a single and simple interface for:

- upload: copy file into HDFS and create basic structure;
- get and search;
- adding new (vertical) data (in general by a MapReduce job);
- create new indices.

These interfaces are implemented in Hadoop by Java classes with remote access and used through a Python client.

Figure 2 shows the internal organization of the core system. The Generator module creates new filesets and registers them into the Catalog. The Creator creates new filesets. The Extender adds more information (new columns). The Appender adds new data. The Indexer creates new indices. The Convertor converts filesets to different formats, including the MapFiles. The Accessor allows the remote access, through an HTTP server on a dedicated machine; a publicly visible WebService can be accessed through the command line or the graphical interface. The Reader performs the search using one of the available tools (Searcher, Scanner or MapReducer), depending on the query.

Figure 3 shows the file organization and relationships in the Hadoop file system (HDFS). The data are stored in collections of "filesets", where each collection represents an event collection (grouping all events that have

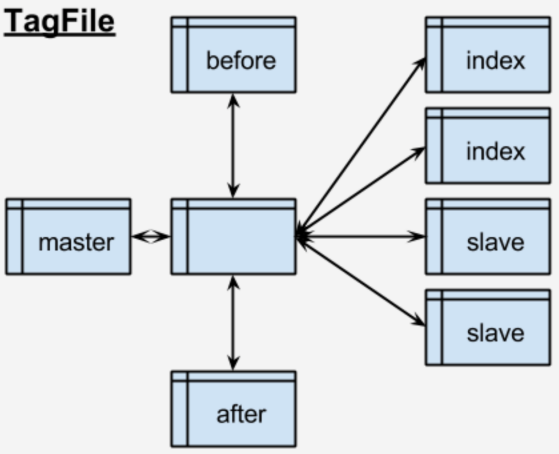

Figure 3: Hadoop file organization.

been processed by the same software version). The TagFiles can be files or directories of files. A collection of TagFiles makes a TagSet, each of which has a master record, pointers to before/after filesets (vertical partitions), slave TagFiles (horizontal partitions) and index TagFiles. The catalogue checks the presence and consistency of all components. Each TagFile is represented by one entry in an HBase [25] table.

8. Query services

The query services are the interface to the core Hadoop system, and solve the proposed use cases for final users. The physicists can use two tools:

1. A command line interface, that provides access to all the functionalities from a console of a controlled environment. Currently the user has to be listed and have explicit access to the Hadoop cluster in order to run correctly all the functions.
2. A Web Service interface, in order to fulfil the Panda Event Service use case. It is a web server accessible from a graphical browser, or from command line tools like wget [26].

An important use case is the query (or count) of the events that satisfied a given trigger condition, or a combination of trigger conditions. In ATLAS, every "fired" trigger condition is marked by a bit set to "1" in the relevant trigger word (Level-1, Level-2 and Event Filter). The correspondence between bits in the trigger words and the actual triggers depends on the run and luminosity block, as the list of active triggers and their possible prescale factors are adjusted according to the variations

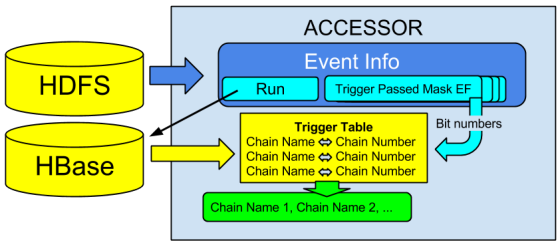

Figure 4: Trigger decoding.

of the LHC luminosity, in order to maximize the collection of useful data for physics analyses. Queries to the EventIndex involving trigger chains need to be first decoded by knowing the trigger menu via the run number, then finding in the COMA database [27] the corresponding trigger name for each bit in the trigger words. The COMA database is stored in Oracle and accessed using ODBC drivers, so the trigger decoding is done beforehand consulting this database, and also cached to improve the future searches with MapReduce jobs in the Hadoop cluster. This process is depicted in Figure 4.

The performance of the data query interfaces is crucial for the usability of the EventIndex. For the initial tests, we imported a full year worth of ATLAS data (2011, first processing version) from the TAGDB into Hadoop and catalogued them. This corresponds to just over 1 TB of data in the CSV (comma-separated values) format, before the internal replication. Simple data scan operations on this dataset using the available Hadoop cluster of 20 nodes take about 15 minutes; we consider this as the worst case, as normally no query would need to read in all data from disk for a full year of data taking. Queries that give the run number and event number (the event picking use case) are very fast; even using the so-far unoptimised "accessor" method these queries return their results in 3.7 seconds (just "fast"). For comparison, the same queries using a full Map-Reduce job return in 90 seconds. A typical query from the Panda Event Service for all events from a file, given the GUID [15] of the file, takes from 3 to 10 seconds. Again, these should be considered as worst cases as they are the results of the first prototype implementation.

9. Outlook

The full chain of the EventIndex project was tested with the first prototype, in different areas. The production of EventIndex information worldwide at different sites was tested using common grid job submission tools, and using the messaging system to convey the in-

formation to the central broker at CERN. The performance showed that it can cope with the required rates for the next few years (LHC Run 2). The Hadoop code was tested with previously loaded 1 TB of 2011 data, and receiving continuously the new data collected by the consumers. The query services provided to satisfy the new use cases show good performance, and still suitable to improve.

Next steps consist in the automation of all the procedures, then loading all Run 1 data, and finally tuning all the system, also providing a complete monitoring system for the final operators. According to the original plan and the current experience, the system will be fully working at the start of 2015, ready for the new LHC data.

References

[1] ATLAS Collaboration, 2005 ATLAS Computing Technical Design Report, ATLAS TDR–017, CERN-LHCC-2005-022, CERN (Geneva, Switzerland), ISBN 92-9083-250-9. https://cdsweb.cern.ch/record/837738/files/lhcc-2005-022.pdf

[2] ATLAS Collaboration, 2008 The ATLAS Experiment at the CERN Large Hadron Collider. Journal of Instrumentation 3 (8): S08003. http://dx.doi.org/10.1088/1748-0221/3/08/S08003

[3] Cranshaw J et al. 2008 Building a scalable event-level metadata service for ATLAS J. Phys. Conf. Ser. 119:072012, http://iopscience.iop.org/1742-6596/119/7/072012

[4] Cranshaw J et al. 2008 Integration of the ATLAS tag database with data management and analysis components J. Phys. Conf. Ser. 119:042008, http://iopscience.iop.org/1742-6596/119/4/042008

[5] Cranshaw J et al. 2008 A data skimming service for locally resident analysis data J. Phys. Conf. Ser. 119:072011, http://iopscience.iop.org/1742-6596/119/7/072011

[6] Mambelli M et al. 2010 Job optimization in ATLAS TAG-based distributed analysis J. Phys. Conf. Ser. 219:072042, http://iopscience.iop.org/1742-6596/219/7/072042

[7] Viegas F et al. 2010 The ATLAS TAGS database distribution and management: operational challenges of a multi-terabyte distributed database J. Phys. Conf. Ser. 219:072058, http://iopscience.iop.org/1742-6596/219/7/072058

[8] Cranshaw J et al. 2010 Event selection services in ATLAS J. Phys. Conf. Ser. 219:042007, http://iopscience.iop.org/1742-6596/219/4/042007

[9] Ehrenfeld W et al., on behalf of the ATLAS Collaboration. 2011 Using TAGs to speed up the ATLAS analysis process J. Phys. Conf. Ser. 331:032007, http://iopscience.iop.org/1742-6596/331/3/032007

[10] Barberis D et al., on behalf of the ATLAS Collaboration. 2014 The future of event-level information repositories, indexing, and selection in ATLAS Proceedings of CHEP 2013, https://cds.cern.ch/record/1625848/files/ATL-SOFT-PROC-2013-046.pdf

[11] Lith, Adam; Jakob Mattson (2010). "Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data", http://publications.lib.chalmers.se/records/fulltext/123839.pdf

[12] Brun R and Rademakers F 1997 ROOT – An Object Oriented Data Analysis Framework Nucl. Inst. & Meth. in Phys. Res. A 389 81-86, http://www.sciencedirect.com/science/article/pii/S016890029700048X

[13] Antcheva I et al. 2009 ROOT – A C++ framework for petabyte data storage, statistical analysis and visualization Comp. Phys. Comm. 180-12 2499–2512, http://www.sciencedirect.com/science/article/pii/S0010465509002550

[14] The PanDA Production and Distributed Analysis System. https://twiki.cern.ch/twiki/bin/view/PanDA/EventServer

[15] A Universally Unique IDentifier (UUID) URN Namespace, http://www.ietf.org/rfc/rfc4122.txt

[16] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino and D. Quarrie, "The athena control framework in production, new developments and lessons learned," *In *Interlaken 2004, Computing in high energy physics and nuclear physics* 456-458*

[17] SQLite3: http://www.sqlite.org

[18] Prun: https://twiki.cern.ch/twiki/bin/view/PanDA/PandaRun

[19] STOMP: http://stomp.github.io

[20] JSON: http://json.org

[21] ActiveMQ: http://activemq.apache.org

[22] HADOOP: http://hadoop.apache.org

[23] HDFS: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[24] T. White, "Hadoop: The Definitive Guide.". O'Reilly Media. 3rd Edition, May 2012. ISBN-13: 978-1449311520

[25] HBase: http://hbase.apache.org

[26] GNU Wget: http://www.gnu.org/software/wget/

[27] Gallas EJ et al. 2012 Conditions and configuration metadata for the ATLAS experiment J. Phys. Conf. Ser. 396: 052033, http://iopscience.iop.org/1742-6596/396/5/052033