HBase/Phoenix-based Data Collection and Storage for the ATLAS EventIndex

Carlos García Montoro[1], *Javier* Sánchez[1], *Dario* Barberis[2,*], *Santiago* González de la Hoz[1], and *Jose* Salt[1,**]

[1]Institut de Física Corpuscular (IFIC), University of Valencia and CSIC, Valencia, Spain
[2]University of Genoa and INFN, Genoa, Italy

Abstract. The ATLAS EventIndex is the global catalogue of all ATLAS real and simulated events. During the LHC long shutdown between Run 2 (2015-2018) and Run 3 (2022-2025) its components were substantially revised, and a new system was deployed for the start of Run 3 in Spring 2022. The new core storage system is based on HBase tables with a Phoenix interface. It allows faster data ingestion rates and scales better than the old system. This paper describes the data collection, the technical design of the core storage, and the properties that make it fast and efficient, namely the compact and optimized design of the events table, which already holds more than 400 billion entries, and all the auxiliary tables, and the EventIndex Supervisor, in charge of orchestrating the whole data collection, now simplified thanks to the Loaders, the Spark jobs that load the data into the new core system. The extractors, in charge of preparing the pieces of data that the loaders will put into the final back-end, have been updated too. The data migration from HDFS to HBase and Phoenix is also described.

1 The ATLAS EventIndex

The ATLAS EventIndex is the catalogue of all real and simulated events collected or produced by the ATLAS experiment [1] at CERN. The LHC Run 2 implementation of the EventIndex is fully described in [2] but since then considerable upgrades have been made, so that the system can continue to provide a stable and reliable service for LHC Run 3 and beyond. A general overview of the architecture, deployment and operation of the ATLAS EventIndex for LHC Run 3 was presented at this conference in Ref. [3].

In this paper we describe how the EventIndex data is collected and stored for LHC Run 3, focussing on the updates made in two very important components: the Data Collection and the Data Storage systems. The implementation of these updates allows us to address the performance challenges of the current run (Run 3) and the future High-Luminosity LHC runs.

*e-mail: Dario.Barberis@cern.ch
**e-mail: Jose.Salt@ific.uv.es

2 Data Collection system

The Data Collection system consists of the processes that bring together the index metadata produced by jobs run at Tier-0 or on the Grid, validates this information, assuring correctness and completeness, and loads the metadata into the EventIndex back-end storage system. So far we collected about 533 billion event records, contained in 32 million indexed files that belong to 280 thousand indexed datasets. Each event record contains the identifiers of this event (run and event number, trigger stream), the trigger information and the pointer to the file containing this event.

The EventIndex components have undergone a series of evolution steps. Initially the Data Collection system was exclusively based on messaging from producers to consumers. This made the system cumbersome and not re-playable. The solution taken was to have a back-end based on Hadoop HDFS files [4] organized per dataset. One of the most important updates has been the implementation of a supervisor process, the Data Collection Supervisor that was introduced to orchestrate the whole data collection procedure. Among the main advantages we have are the accountability of data collection and validation, and an Amazon-S3-like Object Store [5] at CERN that replaced the pure messaging architecture. With these implementations the system is simpler and more scalable and the production can be stored, backed up, and replayed.

The new back-end store is based on Apache HBase [6], which is compatible with the Apache Phoenix [7] SQL interface. This allows improved data structures and simplified management. The consumers are replaced by Spark/Scala Loaders, obtaining a better performance. The result is improved scalability and adaptability, using Open Source industry standards.

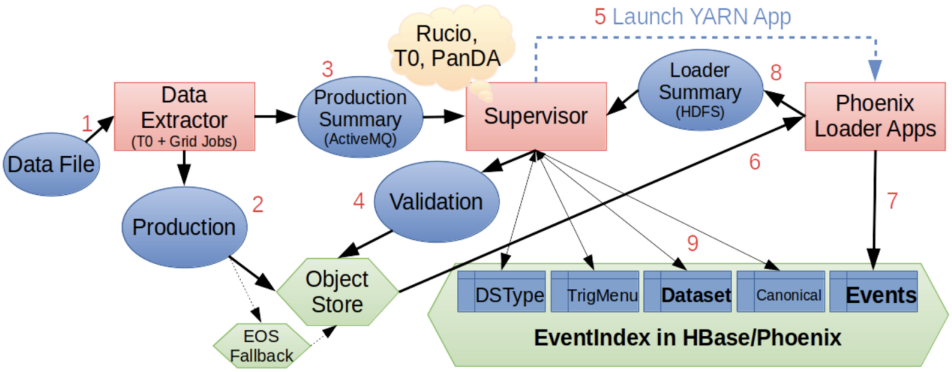

Figure 1. Schema of the data flow within the Data Collection system. The numbers in red mark the different steps in the workflow that are explained in the text.

3 Data Collection workflow

The process of Data Collection can be schematized as displayed in Figure 1. The workflow comprises the following steps:
1 – All data files to be indexed are processed on the Grid or on the CERN Tier-0 cluster by the EventIndex Producer script.

2 – The Producer extracts the event metadata and saves them in a temporary file, which is sent to the CERN Object Store, or to CERN EOS as a fall-back solution. Around 0.1% of transfers to the Object Store fail. Since grid jobs run all over the world on over hundred sites, these failures comprise different causes like site misconfigurations, proxy failures, network problems, etc.
3 – Summary information is sent by each job to CERN through the ActiveMQ [8] messaging server. Potentially, the producer could communicate with the supervisor directly, but we considered more useful to have the CERN ActiveMQ high-availability service as a buffer in case the supervisor is down or overloaded.
4 – When all files belonging to a dataset are processed, a validation step is launched to check the data completeness and consistency, also comparing the event counts with other metadata sources such as the Rucio data management system [9] and the PanDA workflow management system [10].
5 – The Supervisor creates the identifiers needed to identify the dataset in the Phoenix event table, the DSPID (dataset identifier) and the dataTypeId (data type identifier), and creates and queues a new PhoenixLoader object.
6 – The Phoenix thread enforces a policy that limits the number of running loaders and avoids concurrency between loaders with the same DSPID, follows the progress of the applications, validates the result of the application and updates the Phoenix tables.
7 – The PhoenixLoader application is a YARN [11] application in charge of filling the events information of a dataset.
8 – Once completed the loading task, the Loader reports the result to the Supervisor. The PhoenixLoader object handles and records Phoenix Loader application related info, like timing, task, dataset and validation version, status and errors.
9 – At the end of processing the Supervisor updates the other tables.

4 Data Storage logical architecture

The Data Storage system was completely re-implemented in advance of LHC Run 3, after extensive studies covering several ways to store, search and retrieve the EventIndex data

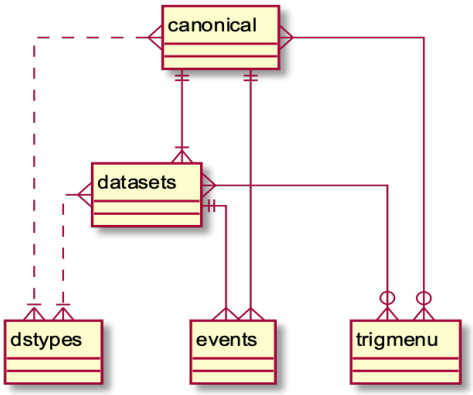

Figure 2. Logical architecture of the Data Storage system, showing the "canonical", "datasets" and "events" tables that keep metadata related respectively to containers, datasets and events, and the auxiliary "dstypes" and "trigmenu" tables with information related to data types and trigger menus.

Figure 3. Table schemas of the Data Storage system implemented in HBase. The top block of each table shows the components of the primary keys, and the other blocks show the contents of each table (divided into column families for the events table) and their representations.

[12]. The logical architecture of the Data Storage system is shown in Figure 2. It follows the organisation of ATLAS data: events are grouped into files that belong to datasets. Datasets are themselves grouped into containers, and this is the unit of data processing for production and analysis tasks.

The EventIndex information in HBase is organised into a "canonical" table that contains metadata related to containers, a "datasets" table with information related to datasets, and the "events" table with event-related information. Two auxiliary tables, "dstypes" and "trigmenu" contain information that can be pointed to from the other tables, respectively on dataset structure and on the trigger menus that are active during different runs.

4.1 Event Record Table

The "events" table is the most important component of the storage system, as it has to contain several hundred billion rows (one per event record), it has to be continuously updated with new data and it has to be searched with good performance. As shown in Figure 3, the first block corresponds to a very optimized primary key. Since the datasets names are too long, generated dataset identifiers have to be used instead. The dspid and dstypeid are generated by the supervisor by means of the autoincremental feature of its RDBMS applied respectively to the dataset name (excluding the data type field) and to the data type string. Monotonically increasing keys are undesirable in HBase, so reversing the bits of dspid populates all the key space uniformly as shown in Figure 4. In this way we achieved a balanced use of all regions and region servers, and the locality of events of each dataset, as each dspid and dstypeid pair identifies a dataset, and the locality of derivations for dataset overlap calculations, as the same dspid and different dstypeid identify related datasets (originating from the same raw data, or simulated events. Seq is a CRC16 to record duplicates, if any, with the first bit reserved to distinguish between data and MC and bits 2 to 4 for versioning.

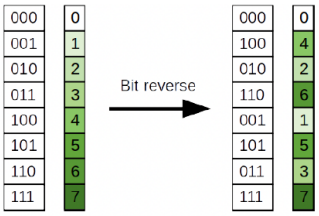

Figure 4. Bit Reverse schema.

Another important feature is that this primary key is compact: it is small, 128 bits, and identifies the dataset with a pair of numbers, no names. Similarly, the Trigger information is stored as smallint arrays.

In other blocks one can find the families to read just what is needed for each use case:

- For event picking in the second block
- For provenance, in the third block
- For Level-1 trigger operations (count, overlap,...) in the fourth block
- For Level-2 and High-Level Trigger operations, in the fifth block

One of the advantages of this structure is that it is compatible with Phoenix, but without using its exclusive features; thus it depends only on HBase, but not on Phoenix.

5 Phoenix Importer and Loader

To write the EventIndex data into the HBase/Phoenix tables, Spark [13] jobs written in the Scala language [14] are used, using the Resilient Distributed Dataset API [15].

The PhoenixImporter could read the data present in the old HDFS implementation of the EventIndex; it was used during the transition to the current storage system to migrate the old data to HBase/Phoenix. The PhoenixLoader can read the EventIndex data from the CERN Object Store (S3). Both the Importer and Loader insert data into the new HBase tables in the same format.

The Phoenix Loader and Importer data flow, shown in Figure 5, consists of the following steps:

1 – Prepare the data source to read EventIndex data files.
2 – Join all files for a dataset into one.
3 – Take the first 10 rows to examine the trigger existence and determine the data format.
4 – Get EventIndex records and decode them into the new schema. Count events, GUIDs, and dstypeids. This step represents most of the work done by the Loader.
5 – Filter null rows that come from records which do not contain event information.
6 – Drop empty columns: trigger, MC channel and weight if non-existent.
7 – Write to Parquet data format [16] if requested.
8 – Reduce the number of partitions.
9 – Write to HBase/Phoenix.

Note that the previous system was based on HDFS MapFiles (comma-separated sequential files), one per dataset. EventIndex files on the Object Store had to be processed and transferred to HDFS by dedicated servers running Java consumers, then a second step was needed to sort all events in a dataset by event number, and a third step to augment the information with trigger names. In the new system based on HBase/Phoenix there is no need to

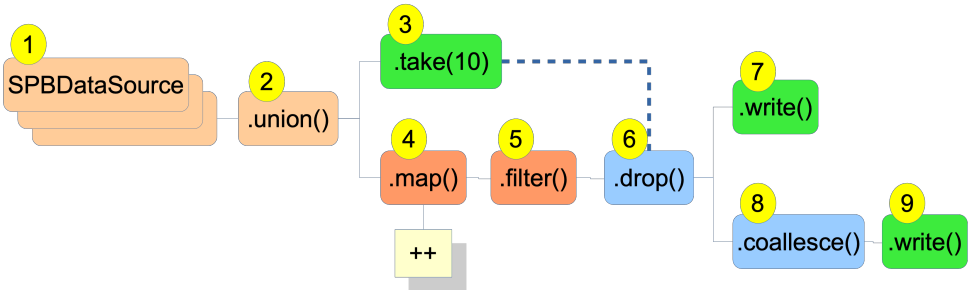

Figure 5. Workflow of the Phoenix Loaders and Importers. The numbers mark the actions taken by each component that are explained in the text.

have private servers running the consumers, since they are replaced by Spark/Scala Loaders that run on the same CERN Hadoop infrastructure. Besides that, events are implicitly sorted by event number (within the dataset) since the key is built with this purpose in mind and trigger information is decoded by the loader itself. Furthermore, the Data Collection became more resilient because of the use of Resilient Distributed Dataset by the new Spark EventIndex Loader, so data ingestion into HBase has more probability to succeed in situations were the Object Store or HBase have problems.

6 Conclusions

Major updates have been applied to the EventIndex Data Storage system in order to improve its performance for LHC Run 3. The main store is now implemented as a set of very large but very optimized HBase tables with a Phoenix interface for SQL commands.

As a result, the Data Collection component is also working reliably, including Producers, Supervisor, Loaders, and Importers. Significant work has been invested in setting up and running an automatic, self-managed, self-regulated and fully accountable data migration procedure from HDFS to HBase/Phoenix, and setting up and running the regular data collection for HBase/Phoenix.

The Spark data processing paradigm and the Scala language allowed us to design a simpler and more robust data loading model in HBase/Phoenix than was possible with the previous data storage implementation in HDFS MapFiles. Both the PhoenixLoader and the PhoenixImporter have been running in stable production with no issues since spring 2022. They are being used to populate the production HBase/Phoenix database.

This work was partially supported by MICINN in Spain under grant PID2019-104301RB-C21.

References

[1] ATLAS Collaboration, *The ATLAS experiment at the CERN Large Hadron Collider*, *JINST* **3** (2008) S08003. https://doi.org/10.1088/1748-0221/3/08/S08003

[2] Barberis D. et al., *The ATLAS EventIndex: A BigData Catalogue for All ATLAS Experiment Events*, *Comput.Softw.Big Sci.* **7** (2023) 1,2. https://doi.org/10.1007/s41781-023-00096-8

[3] Gallas E. J., et al., "Deployment and Operation of the ATLAS EventIndex for LHC Run 3", these proceedings (2023).

[4] Apache Hadoop and associated tools: https://hadoop.apache.org

[5] Fernández Casaní A. et al., *A reliable large distributed object store based platform for collecting event metadata*, *J Grid Comp* (2021) 19:39. https://doi.org/10.1007/s10723-021-09580-0

[6] Apache HBase: https://hbase.apache.org

[7] Apache Phoenix: https://phoenix.apache.org

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

[9] Barisits M. et al., *Rucio: Scientific Data Management*, *Comput.Softw.Big Sci.* **3** (2019) 11. https://doi.org/10.1007/s41781-019-0026-3

[10] Barreiro Megino F.H. et al., *PanDA for ATLAS distributed computing in the next decade*, *J.Phys.Conf.Ser.* **898**:052002 (2017). https://doi.org/10.1088/1742-6596/898/5/052002

[11] Apache Hadoop YARN: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

[12] Barberis D. et al., *The ATLAS EventIndex for LHC Run 3*, *EPJ Web of Conferences* **245**, 04017 (2020). https://doi.org/10.1051/epjconf/202024504017

[13] Apache Spark: https://spark.apache.org

[14] Scala programming language: https://www.scala-lang.org

[15] Resilient Distributed Dataset API: https://www.sciencedirect.com/topics/computer-science/resilient-distributed-dataset

[16] Parquet data format: https://parquet.apache.org