EPJ Web of Conferences **214**, 04010 (2019) https://doi.org/10.1051/epjconf/201921404010
CHEP 2018

Distributed Data Collection for the Next Generation ATLAS EventIndex Project *

Álvaro Fernández Casaní[1,**], *Dario* Barberis[2], *Javier* Sánchez[1], *Carlos* García Montoro[1], *Santiago* González de la Hoz[1], and *Jose* Salt[1] on behalf of the ATLAS Collaboration

[1]Instituto de Física Corpuscular (IFIC), Universidad de Valencia and CSIC
[2]Università e INFN Genova

Abstract. The ATLAS EventIndex currently runs in production in order to build a complete catalogue of events for experiments with large amounts of data. The current approach is to index all final produced data files at CERN Tier0, and at hundreds of grid sites, with a distributed data collection architecture using Object Stores to temporarily maintain the conveyed information, with references to them sent with a Messaging System. The final backend of all the indexed data is a central Hadoop infrastructure at CERN; an Oracle relational database is used for faster access to a subset of this information. In the future of ATLAS, instead of files, the event should be the atomic information unit for metadata, in order to accommodate future data processing and storage technologies. Files will no longer be static quantities, possibly dynamically aggregating data, and also allowing event-level granularity processing in heavily parallel computing environments. It also simplifies the handling of loss and or extension of data. In this sense the EventIndex may evolve towards a generalized whiteboard, with the ability to build collections and virtual datasets for end users. This proceedings describes the current Distributed Data Collection Architecture of the ATLAS EventIndex project, with details of the Producer, Consumer and Supervisor entities, and the protocol and information temporarily stored in the ObjectStore. It also shows the data flow rates and performance achieved since the new Object Store as temporary store approach was put in production in July 2017. We review the challenges imposed by the expected increasing rates that will reach 35 billion new real events per year in Run 3, and 100 billion new real events per year in Run 4. For simulated events the numbers are even higher, with 100 billion events/year in run 3, and 300 billion events/year in run 4. We also outline the challenges we face in order to accommodate future use cases in the EventIndex.

1 Introduction

The ATLAS EventIndex [1] is a catalog of real and simulated events, in all processing stages, for satisfying several use cases. A small quantity of metadata per event is indexed and collected worldwide, including event identifiers (run and event numbers, trigger stream, lumi-

*This work was partially supported by MICINN in Spain under grants FPA2016-75141-C2-1-R, which include FEDER funds from the European Union.

**e-mail: Alvaro.Fernandez@ific.uv.es

nosity block), the online trigger pattern (L1,L2,EF/HLT), and references to the files that contain the event in the different processing step. Users can select single events from billions of entries depending on selected constraints. Automatic systems can do production consistency checks, including duplicate event checkings across different files and datasets, or compute the overlaps in derivation framework detecting common events across different data files. Other analytic use cases include counting or give an event list based on a trigger selection.

2 Distributed Data Collection Architecture

The Data Collection task is in charge of collecting all indexed data and forward it to the Data Storage Services. The current distributed producer/consumer architecture based on a Object Store was defined in 2016 [2] and is now being used in production since July 2017. In this architecture, as depicted in Figure 1, Producers run at CERN Tier-0 and at Grid sites worldwide, indexing input data files with an Athena Python transformation, and storing this index in an Object Store. In addition small control messages are sent to a supervisor with the ActiveMQ messaging Server. The Supervisor runs centrally at CERN, and controls all the processes and validates the indexed data at the desired granularity, currently the dataset level. It notifies the Consumers about valid unique data ready to be ingested to Hadoop HDFS, where the data is finally available to users and other systems. A web application provides information about all the operations and the current state of the Supervisor and the data collection process.

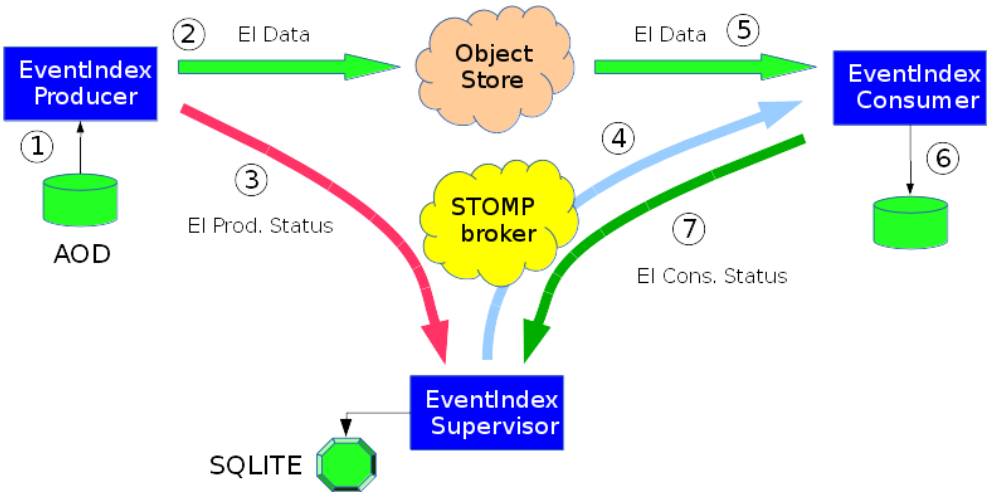

Figure 1: Distributed Data Collection Architecture

2.1 Performance and results

Since the Object Store method was put in production, almost 2 million Producer jobs where run at CERN Tier-0 and submitted to over 100 sites distributed worldwide. The index data is collected by our system, but also other statistics per Producer like its identifiers (including Tier-0 or Panda job Id, and task and attempt numbers), the start and end execution time, the input dataset name and the number of input files, and the number of indexed events and the output size of the index data (including the raw size and the actual size as stored in the Object

Store). These control messages and statistics are received and maintained in a local database by the Supervisor, which then we can use to produce the numbers and graphics presented in the following subsections, which summarizes one year of operation and statistics collection.

2.1.1 Event processing rates

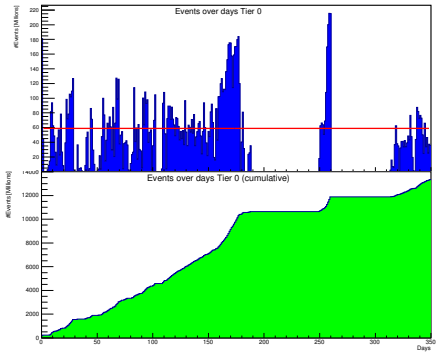

Figure 2: Tier-0 processed events

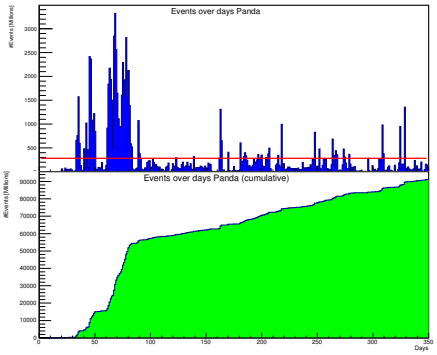

Figure 3: Grid processed events

The main indicator that the overall distributed data collection system is performing correctly is throughput, measured in the rate of indexed events. In the following Figures we can see the number of events indexed and stored in our system during last year. Figure 2 shows the number of events indexed per day at CERN Tier-0, while Figure 3 represents the number of events indexed per day at all ATLAS grid sites. The x-axis represents time in number of days since the start, so basically 1 year (350 days) of data taking. The y-axis represents the number of events indexed, in millions. Each figure is divided in two frames, so we can see the blue bars in the upper part of the figure that represent the number of events indexed each day, and in the lower part of the figure we can see in green the accumulated number of events until that day. The red line in Figure 2 represents the mean, accounting for around 60 million events indexed per day at CERN Tier-0. There are periods of time when there are no jobs running at Tier-0 for different reasons (no production, maintenance, etc), and others with higher production peaks, reaching 220 million events processed in one day. The green filled graph provides the total number of events indexed at the day 350 (June 2018) which is over 13.5 billion events. Correspondingly, in the Figure 2 representing the Grid production, we can see the mean of indexed events is higher, about 280 million events every day on the ATLAS Grid sites. There are peaks of 3500 million events in a single day, and the total indexed over last year was about 91 billion events. The totals summing CERN Tier-0 and all Grid sites account for 105 billion events indexed during last year with the new Object Store system, with peaks of 3.5 billion events indexed in a single day.

2.1.2 Object Store rates

The CERN Object Store based on Ceph[3] is used as intermediary storage where the Producers submit their payload when they finish the indexing phase of one or more input files. From there, Consumers can get the intermediate results to consolidate them at the CERN Hadoop instance at the desired granularity, usually at the dataset level. Therefore the Object Store is

an important component and in this section we will show the object creation rates and amount of space that we are using. Figure 4 shows the object creation rate by the Producers and the red line representing the mean of 5.6k objects created per day, with peaks of 44k objects created in one day. A total of 2 million objects were created so far. In Figure 5 we show the objects size, so the y-axis represents the amount of data in objects created (in GiB) during a particular day in the blue bars, and the mean of data created per day is 15 GiB, represented by the red line. The total amount of data created on CERN Ceph was over 5 TiB at the end of last day, corresponding to June 2018. It must be noted that the data is a factor 10 compressed related to the original event index information. Figure 6 represents a histogram of the object size, so we can obtain more information on how the amount of data is distributed among the different stored objects. The x-axis represents the size of the object (in MiB) and the y-axis the frequency or the number of occurrences. The mean object size is 2.6 MiB, and 99 % of the objects are less than 15 MiB. There are some bigger objects, being the biggest one 670 MiB in size.

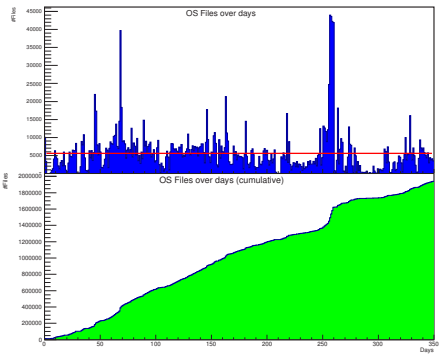

Figure 4: Object creation in time

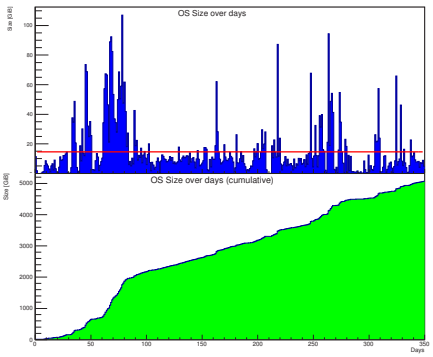

Figure 5: Object size in time

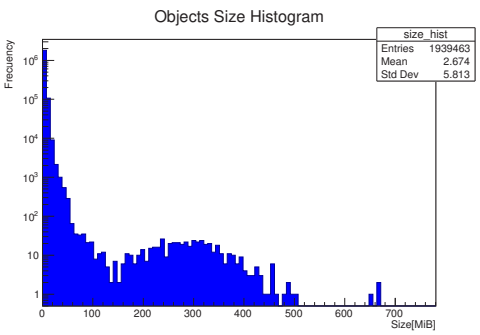

Figure 6: Objects size histogram

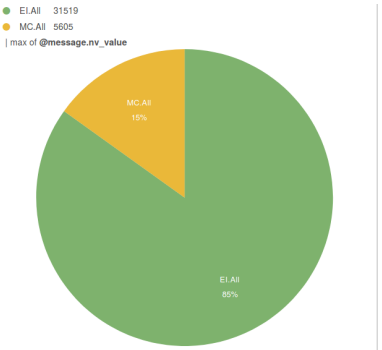

Figure 7: Hadoop(HDFS) storage usage

2.1.3 Ingestion rates

Ingestion is performed by the Consumers, that read the information from one or more valid objects from the Object Store and consolidate final data in the Hadoop HDFS Storage. The

granularity of the ingestion is determined by the Supervisor, which currently waits until a complete dataset is indexed and temporarily stored at the Object Store. At this point the Supervisor signals the Consumers which objects contains unique valid data. Currently around 40% of the datasets are contained in a single object. Most of the datasets occupy less than 75 MiB, and the biggest one comprises 8000 objects and around 7 GiB. A single consumer can ingest in the order of 15k events/s and we can scale horizontally increasing the number of Consumers. The data is consolidated at the dataset level writing a single HDFS file per dataset in a directory named after the container. This reduces the total number of written files compared with the previous approach. In Figure 7 we can see the current event index data stored in Hadoop HDFS, summing up the data indexed for all campaigns. It contains 37 TiB of indexed events data in Hadoop in a compressed mapfile format, which corresponds to the original 167 TB before compression. From this data, 31 TiB correspond to real data, and 6 TiB from Montecarlo simulated data.

3 Evolution for the Next Generation EventIndex

During this year 2018 the recording rate for all ATLAS processes was measured up to 30 billion events/day (up to 350 Hz on average). In the future Run-3 that starts in 2021 there will be an increase of trigger rates, so we need to scale at least half an order of magnitude. Beyond the future increasing demanding rates, there are other guidelines that will drive the evolution of the EventIndex. The first driver is improving the current implementation approach. One issue in this area is how different incarnations of the same event in each processing step (RAW, ESD, AOD, DAOS, NTUP) is represented in the EventIndex. Currently every reprocessing is physically stored in different Hadoop HDFS files. This is due to the organization of storage by dataset name, that in fact contains the processing step type. A guideline is to explore approaches and technologies that ease the procedure of maintaining one and only one logical record per event. A second driver for the evolution of the EventIndex is the inclusion of new functionality. In this area we are aiming to include support for Virtual Datasets, created either explicitly (providing a collection of event identifiers) or implicitly (a selection based on some other collection or event attributes). In addition it would be useful to have the possibility to label individual events with key-value attributes, either by a process or by a final user.

3.1 New Backend solution: Kudu

Searching new solutions for the optimization and unification of data storage for the different use cases of the EventIndex, we have first decided to explore new backend solutions. Apache Kudu[4] is a new columnar-based storage that allows fast insertions and retrieval, and fills the gap of applicability between HDFS and HBase. In terms of the pace of Data, HDFS is designed for storing unchanging of append-only data, but Kudu allows frecuent updates and fast changing data, without reaching HBase performance on real-time changing updates. In addition Kudu allows Fast analytics ouperforming HBase, without reaching HDFS hability to do fast scans on static data only. In particular for our EventIndex application, we would benefit from two main characteristics. First, the unification of the data for all use cases which is currently not possible in our implementation. Currently, all EventIndex data is stored in HDFS, which provides a reliable and safe method to maintain them. Use cases that involve searching across a large amount of data are done in a performant way. This includes searching for event duplicates, calculate the overlap matrix of events or specific triggers across different dataset derivations, or any other analytic workload. On the other hand, HDFS is not well suited for use cases that involve fast access to a small and defined amount of data. For example for event picking, or selecting the location of a particular event among billions,

we should tackle the problem with a different approach. In this case a small quantity of data is replicated in HBase, including the event identification and the GUID of the file in the grid. HBase is performant enough to provide real-time access to such information. The disadvantage is the non-negligible amount of replicated data, and the cost in the management of the life cycle of this replicated data, including the copy and deletion of no-longer valid data to be synchronized with the main data in the HDFS repository. In addition more data is copied outside Hadoop to Oracle, to provide fast dataset discovery. With Kudu we aim to unify the data store to solve the random access and analytic use cases, effectively reducing the amount of total data stored and avoiding complex and error-prone synchronization procedures. The second benefit would arise from the fact that in Kudu, we can make related data (reprocessing of the same events) sit close to each other on disk. On Kudu, the data organization is done in tables that have a defined schema, with primary keys and partitions defined with a subset of the keys. It is not possible to define foreign keys like in traditional relational databases, but this is not a constraint for our use cases. The ingestion and query scanning are distributed amontg the servers holding the partitions, called tablets. This improves performance and alsoavailability, as data is also replicated in several tablets on different servers. A careful key and partition schema definition for the EventIndex is being designed[5], with the aim to allow that row entries that define different reprocessings of the same data sit in the same partition and close in disk storage. This fact improves navigation through locality, and also benefits from better compression ratios of the same data due to this locality. More details about the implementation on Kudu and the tests that we have been performing related with the Data Collection task can be seen in Ref. [5].

4 Summary

The EventIndex Distributed Data Collection is currently running in production for the Run-2 of the ATLAS Experiment, indexing and collecting billions of events worldwide. The system was redesigned and implemented after some congestion problems detected with the messaging system, that could compromise scaling up the data rates for future years. The new Object Store based implementation was put in production in 2017, and during last year we have indexed 300 million events per day. We have improved the system in other areas compared with the previous implementation. Producer payloads can now be encoded in a single object, that is temporary stored in the Object Store for consumption. The current binary data encoding is more compact, and can reduce the amount of data up to a factor 10. This reduces the amount of data transported, temporarily stored, and finally consolidated in Hadoop HDFS. This possibility, in addition of the compression of the previously loaded data into HDFS, has meant a reduction of the space used within the EventIndex project. The Supervisor can select unique validated data, without consuming duplicate data into HDFS and reducing the need of extra cleaning steps on the Hadoop cluster. The consumption process has also seen improved performance, with no blockings detected as it was occurring with the messaging implementation in some situations. There are future challenges regarding the increasing production rates, and the use cases to be solved. The current work is focused on new storage technologies to support fast insertion, and to reduce data replication, unifying low latency random access, and analytic use cases. We performed first ingestion test on Kudu with good results, suggesting that it is a promising backend for the future evolution of the EventIndex project.

References

[1] D. Barberis, S.C. Zárate, J. Cranshaw, A. Favareto, Á.F. Casaní, E. Gallas, C. Glasman, S.G. De La Hoz, J. Hřivnáč, D. Malon et al., *The ATLAS EventIndex: architecture, design choices, deployment and first operation experience*, in *Journal of Physics: Conference Series* (IOP Publishing, 2015), Vol. 664, p. 042003

[2] Á. Fernández Casaní, F. Prokoshin, R. Yuan, J. Sánchez, J. Salt, D. Barberis, R. Többicke, A. Favareto, J. Hřivnáč, C.G. Montoro et al., *ATLAS EventIndex general dataflow and monitoring infrastructure*, in *Journal of Physics: Conference Series* (2017), Vol. 898, p. 062010

[3] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, C. Maltzahn, *Ceph: A Scalable, High-performance Distributed File System*, in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (USENIX Association, Berkeley, CA, USA, 2006), OSDI '06, pp. 307–320, ISBN 1-931971-47-1, `http://dl.acm.org/citation.cfm?id=1298455.1298485`

[4] T. Lipcon, D. Alves, D. Burkert, J.D. Cryans, A. Dembo, M. Percy, S. Rus, D. Wang, M. Bertozzi, C.P. McCabe et al., *Kudu: storage for fast analytics on fast data* (2015)

[5] Z. Baranowski, D. Barberis, L. Canali, A. Fernandez Casani, E. Gallas, C. Garcia Montoro, S. Gonzalez de la Hoz, J. Hrivnac, F. Prokoshin, G. Rybkin et al. (ATLAS Collaboration), *A prototype for the evolution of the ATLAS EventIndex based on Apache Kudu storage*, in *Proceedings of the 23rd International Conference on Computing in High Energy and Nuclear Physics* (EDP Sciences, Les Ulis, France, 2018)

[6] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi et al., *Impala: A modern, open-source sql engine for hadoop*, in *In Proc. CIDR'15* (2015)

[7] K. Masui, *Bitshuffle*, https://github.com/kiyo-masui/bitshuffle (2018)