Big Data Tools as Applied to ATLAS Event Data

I Vukotic[1], R W Gardner and L A Bryant

University of Chicago, 5620 S Ellis Ave. Chicago IL 60637, USA

ivukotic@uchicago.edu

Abstract. Big Data technologies have proven to be very useful for storage, processing and visualization of derived metrics associated with ATLAS distributed computing (ADC) services. Logfiles, database records, and metadata from a diversity of systems have been aggregated and indexed to create an analytics platform for ATLAS ADC operations analysis. Dashboards, wide area data access cost metrics, user analysis patterns, and resource utilization efficiency charts are produced flexibly through queries against a powerful analytics cluster. Here we explore whether these techniques and associated analytics ecosystem can be applied to add new modes of open, quick, and pervasive access to ATLAS event data. Such modes would simplify access and broaden the reach of ATLAS public data to new communities of users. An ability to efficiently store, filter, search and deliver ATLAS data at the event and/or sub-event level in a widely supported format would enable or significantly simplify usage of machine learning environments and tools like Spark, Jupyter, R, SciPy, Caffe, TensorFlow, etc. Machine learning challenges such as the Higgs Boson Machine Learning Challenge, the Tracking challenge, Event viewers (VP1, ATLANTIS, ATLASrift), and still to be developed educational and outreach tools would be able to access the data through a simple REST API. In this preliminary investigation we focus on derived xAOD data sets. These are much smaller than the primary xAODs having containers, variables, and events of interest to a particular analysis. Being encouraged with the performance of Elasticsearch for the ADC analytics platform, we developed an algorithm for indexing derived xAOD event data. We have made an appropriate document mapping and have imported a full set of standard model W/Z datasets. We compare the disk space efficiency of this approach to that of standard ROOT files, the performance in simple cut flow type of data analysis, and will present preliminary results on its scaling characteristics with different numbers of clients, query complexity, and size of the data retrieved.

ATL-SOFT-PROC-2017-001
03 January 2017

1. Introduction

The ATLAS experiment at the CERN LHC [1], as practically all other High Energy Physics experiments, stores collected event data in files for further processing and analysis. As is common in the field, the files are stored in a ROOT [2] format, and in order to optimize disk space, data access rate, and user's experience, custom converters are used to read/write ATLAS data. This limits resources to those systems that have deployed both ROOT and the ATLAS analysis framework. Millions of the event data files produced are distributed around the world for safekeeping and processing by the ATLAS distributed computing (ADC) system. Actual data processing is mostly

[1] Speaker, on behalf of the ATLAS collaboration

done on the Worldwide LHC Computing Grid (WLCG), orchestrated by the Panda job brokering system [3]. The ADC system by itself produces a large amount of operational and metadata that is constantly analyzed in order to monitor and optimize the system. Unlike the event data, the ADC data storage is a combination of relational (Oracle) and modern unstructured data processing systems (Hadoop [4], HBase, Elasticsearch [5]). An analytics system for the ADC provides a very fast, efficient, and open access to large datasets. While there were no efforts to use similar system for the event data, an EventIndex [6] stores event level metadata that is used by the EventService [7] to quickly find and stream files containing actual event data. We tested feasibility of the event data storage in a NoSQL database, described and measured performance of the different modes of data access.

2. Analytics Platform

The workflow we selected to test is shown in Figure 1. In this schema we index ATLAS event data in an Elasticsearch cluster that is then used for all the data accesses. The cluster provides powerful search, filter, and aggregation functionality, accessible via a well documented REST interface. This interface can be used by external clients, both directly or via provided Python, Java and Ruby APIs.

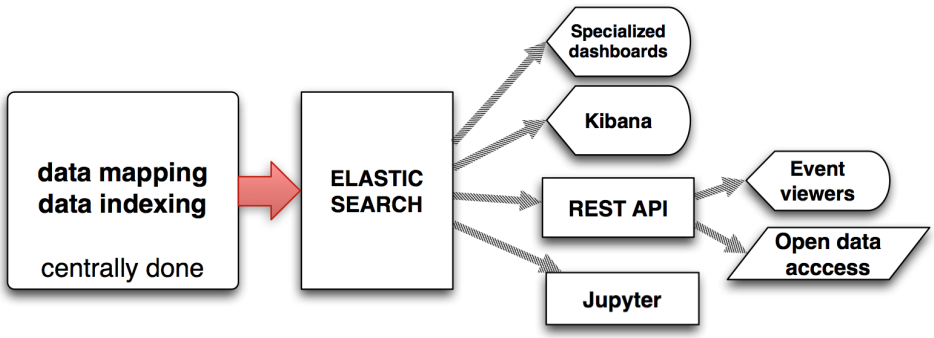

Figure 1. A simplified schema of the proposed way to store and provide access to a part of the ATLAS event data.

Most simple data visualizations can be performed via a web based Elasticsearch GUI plugin, Kibana [8]. While not made with physicists in mind (only a limited set of visualizations and customizations, no print quality output, etc.) it does provide a simple and fast data investigation possibilities with no coding required. The visualizations can be saved, organized in dashboards and shared. For complex analysis tasks, we provide a Jupyter [9] hub node local to the Elasticsearch cluster. This improves performance of queries returning a large amount of data. Physicists used to ROOT can utilize pyroot. It is also possible to use standard machine learning packages: numpy, scikit-learn, Keras (with Theano backend) and matplotlib for complex visualizations. All the development and performance tests quoted below are performed using Elasticsearch cluster (3 master nodes, 5 data nodes with 4TB of SSD storage each) and one Jupyter instance at the University of Chicago.

3. Event Indexing

3.1. ATLAS data formats

While all the ATLAS data is stored in ROOT files, several different formats exist that differ in their content, internal structure, and usage patterns. Here we will not discuss RDO (raw data objects) and ESD (event summary data) formats as these are rarely used in analysis, are rather large, and much costlier to process (RAW data exists only on tape and ESD is transient - to create it one has to re-reconstruct RAW data). We will not discuss the user specific formats (ordinary ntuples) as these are usually of no general interest and are relatively small and fast to process. An analysis is usually performed on Analysis Object Data (AOD) files. The primary AOD data are still too large for most use cases, so the derived AOD (DAOD) files are centrally produced, usually one set per physics

working group. These are skimmed (a subset of events is kept) and slimmed (subset of branches is kept) AODs to roughly 1-5% of their original size, and have all the re-reconstructions, re-calibrations, and corrections already applied. As it can be seen in Table 1, AODs and DAODs significantly differ in both size per event and number of branches.

Table 1. ATLAS data formats. All the data based on one DAOD_EXOT2 dataset.

Type	Event size [kb/event]	Number of Branches
DAOD data15_13TeV	30	647
AOD data15_13TeV	184	1742
DAOD mc15_13TeV	25	999
AOD mc15_13TeV	216	2512

3.2. Event mapping

To index the event data we first need to transform it into a JSON format natively used by the Elasticsearch. Basically we need to create a map of event content to Elasticsearch documents. The mapping is made for each data type separately. It describes how to cast c++ base types to Elasticsearch types, tunes parsing options (e.g. are the string variables split or not), organizes parent-child relationships inside the document, and ensures nested objects are indexed as subdocuments. This proved to be the most difficult task in this project as one has to ensure that search and filter operations will behave in an expected way, while obeying internal Elasticsearch limits (e.g. at most 50 subdocuments per document). It also influences document storage size and search performance.

To generate the mapping templates and later index ATLAS event data, we wrote a Python code and used root_numpy and pyroot packages. This approach is geared towards simplicity and not performance, there are projects in development that aim to provide high performance conversion of ROOT format files into a wide range of industry standard formats (avro, parquet, HDF5, JSON). A snippet of the template used and the data structures involved are shown in Figure 2.

Table 2. Event indexing performance

	Event size		
Type	**[kb/event]**	**increase**	**Indexing rate [Hz]**
DAOD data15_13TeV	146	5.4	21
AOD data15_13TeV	377	2.0	8
DAOD mc15_13TeV	116	4.6	19
AOD mc15_13TeV	385	1.8	9

For this initial testing we choose not to optimize aspects of the mapping that would be dependent on a data use case. For example we index all the variables (TBranches) even though we could index only those that could be reasonably used for search and simply store the rest. This provides the worst case scenario data size.

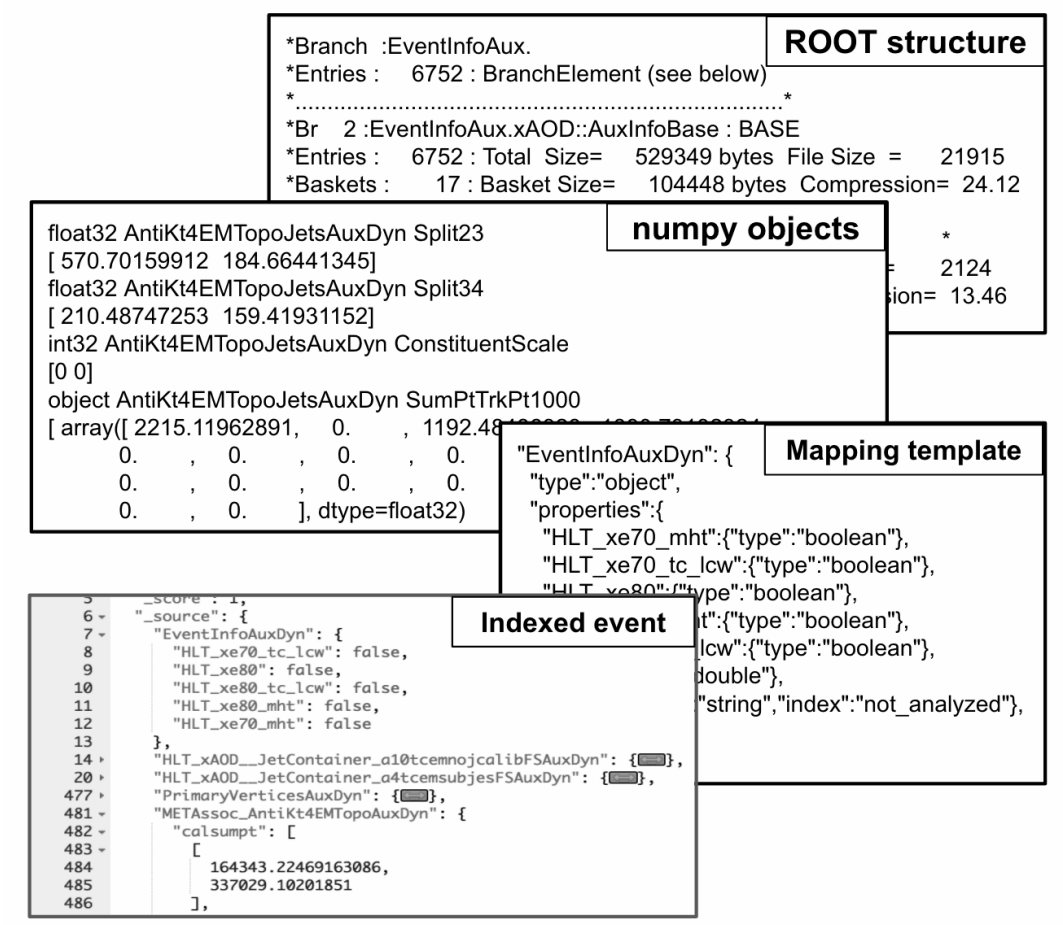

Figure 2. Data transformation from ROOT files to Elasticsearch documents. ROOT files are written with full split option so every variable has its own branch. The branches containing vectors are converted to numpy arrays. The mapping template leaves simple arrays as simple Elasticsearch types, but converts arrays of arrays into sub-documents.

4. Tests and results

There are many scenarios of data access but from the data storage and delivery point of view they differ in how sparse accesses are in terms of rows (event skipping) and columns (variables not used in analysis). Our tests tried to measure performance of the most extreme versions of what a physics analysis would do. The major classes of accesses are: a) where result of analysis depends only on the variables indexed or their relatively simple transformation, and b) when a complex calculation is needed on all or part of variables.

An investigative analysis (e.g. comparing distributions of the Monte Carlo generated and the experimental data) and a cut-flow type analysis (e.g. calculation of trigger efficiencies for a certain class of events) are examples of the former. In these cases almost all of the analysis would be done on the Elasticsearch cluster nodes, amount of data returned would be very small, and it would not contain actual indexed data. Here are a few simple examples together with a corresponding search query:

- simple counting: `"aggs":{"docs":{"value_count":{"field":"_type"}}}`
- simple statistics: `"aggs":{"ptStats":{"stats":{"field":"TrackPt"}}}`

- statistics in bins:
  `"aggs":{"ptStats":{"histogram":{"field":"TrackPt","Interval":50}}}`

The analysis cases that can not be done on the cluster itself (e.g. event mixing for background estimation, application of a machine learning method, etc.) would require an access to the whole or a part of the indexed data. These queries return much more data and that creates stress points at the network link to the Elasticsearch cluster, network link at the client side, and client side CPU utilization in order to parse large number of potentially large JSON documents. Here are two simple examples:

- streaming all events in their entirety: `"query" : { "match_all": {} }`
- to stream few variables simply add: `"fields" : ["AntiKt4EMJet.Area", ...],`

Both analysis cases greatly benefit from being able to do quick filters (cuts in HEP parlance). Elasticsearch supports simple single variable filters, complex boolean expressions, filters based on weights, pseudo random selections and more. Here are two example queries:

- simple one variable filter: `"query" : {"term":{"EventInfoAuxDyn.HLT_xe80":True}}`
- complex logical expression.

```
'bool':{
   'must':[
       {"term" : { "EventInfoAuxDyn.HLT_xe80" : False }},
       {"range": { "AntiKt4EMTopo.AntiKt2TrackJetCount" : {"gt":2}}},
       {"range": { "AntiKt4LCTopo.JetEMScaleMomentum": {"gt":10000}}},
       {"exists":{ "field":"HLT_xAOD__JetContainer.HECQuality"}}
   ]}
```

In our tests of the investigative analysis type, most queries were answered in under few seconds, which is more that fast enough for all the practical purposes. With so short turn-around times it is expected that even a relatively large number of users would not see significant performance loss. An increase in the amount of data stored/processed is contingent to adding new Elasticsearch nodes, so parallel increase in CPU resources would prevent performance degradation. As the Kibana user interface uses the same kind of queries to create visualizations, no performance loss is expected there either.

The results of our tests of the “streaming” analysis type are summarized in Table 2. All the tests were run as Jupyter notebooks on the node described above and in the local network of the Elasticsearch cluster. The event processing rate is consistently higher for DAOD than AOD files. Since AOD events are larger, this result is expected for all the tests reading full events. While the difference in rate is smaller in tests reading the same number of variables, reading from DAOD is still approximately factor of 2 faster than from AOD. This leads us to conclude that there is an extra penalty in having very complex events schema. Indeed Elasticsearch version 5.0 limits number of fields per record to one thousand. Comparing across rows it seems that the less information is returned by the query the higher processing rate will be. This holds true even when comparing queries returning data from all events to the ones returning data from 2% events. We assume this is an effect of the in-memory caching of the search data. All the rates are at the levels that could be expected from a native c++ ROOT code reading ROOT files. The big surprise was a high CPU load observed that we can only attribute to parsing of the returned JSON data. This also varies with amount of data returned, from almost negligible when returning 1-10 variables to very high when streaming full event data. A future study could compare different Python JSON libraries as large variations in their performance have been reported.

Table 2. Performance in different data access scenarios. For tests not reading all events we quote rate as number of events that are delivered not number of events processed. CPU percentage quoted represent a Python client running on a single core, single process and includes parsing JSON data into Python structures.

Read test	DAOD data15_13TeV		AOD data15_13TeV		DAOD mc15_13TeV		AOD mc15_13TeV	
	Rate [Hz]	CPU [%]	Rate [Hz]	CPU [%]	Rate [Hz]	CPU [%]	Rate [Hz]	CPU [%]
one variable from all events	486	1	264	1	520	1	225	1
ten variables from all events	453	3	263	1	491	3	227	1
ten variables from events passing cut (~2 % events)	606	22	251	10	971	15	226	20
full events passing cut (~2% events)	161	77	35	84	121	80	32	82
streaming all events in full	116	86	39	87	120	88	31	86

Most of the full data streaming accesses would be for the purpose of small fixes of the data itself (e.g. recalibrations). These could be done by simply reindexing of the data, or adding columns with new variables and can be done in a batch mode on the cluster itself.

5. Conclusions

There are numerous ways to combine and use modern big data tools to store and analyze HEP event data. We presented our experience with one of them, based on the currently most popular search engine and ATLAS event data. We found that the issue of the data size increase, while significant, would still be acceptable for the DAOD data. The issue could be partly ameliorated by optimizing indexing (e.g. not indexing everything). Analyses that don't need a large fraction of data and complex processing show high event rates and would benefit from almost real time access to the data and simple online visualizations. Complex analyses or analyses that need most of the data show reasonable event rates, and are limited by network bandwidth to client and high CPU cost of parsing JSON format. Both issues could be ameliorated by using compression for the data transfer and using faster JSON parsers. If deployed at scale, advantages of this approach are: simplified data management, better resource utilization, faster turnaround times for most data analysis, and it makes it possible to open the data to outside access in practice and not only in words. It also ensures that everyone uses the same latest data for the analysis, relieves users from data managing issues, batch processing, event accounting. The disadvantages are: our data centers have mainly network attached storage that is very unsuitable for search engine clusters, current data format is not optimized for Elasticsearch, it would significantly increase our disk requirements, and would require a change of data production and management procedures. For an end-user main disadvantage would be a need to learn filtering syntax and rewrite the part of the analysis code responsible for getting the data. We feel that the results obtained warrant further research in this direction.

References

[1] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider* , JINST 3 13617 (2008) S08003

[2] Brun R. and Rademakers F., ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep.1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.

[3] Maeno T., *PanDA: distributed production and distributed analysis system for ATLAS*, Journal of Physics: Conference Series **0119** (062036) 006 [iopscience.iop.org/1742-6596/119/6/062036]

[4] Apache Software Foundation. Hadoop. Jun 19, 2016. URL: https://hadoop.apache.org

[5] Elasticsearch. Open Source, Distributed, RESTful Search Engine https://www.elastic.co/

[6] Barberis D., et al., *The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of dat*a, Journal of Physics: Conference Series **513** (2014) 042002 doi:10.1088/1742-6596/513/4/042002

[7] Calafiura P., et al., *The ATLAS Event Service: A new approach to event processing*, Journal of Physics: Conference Series **664** (2015) 062065 doi:10.1088/1742-6596/664/6/062065

[8] Elasticsearch. Kibana. July 21, 2016. URL: https://www.elastic.co/products/kibana

[9] Project Jupyter. Jupyter. Dec. 04, 2016. URL: http://jupyter.org/