Big Data Analytics Tools

Applied to ATLAS Event Data

Ilija Vukotic
University of Chicago

CHEP 2016, San Francisco

Idea

Big Data technologies have proven to be very useful for storage, visualization and analysis of the ATLAS distributed computing (meta)data. If we could apply the same technologies on real/MC data, we could:

- Do quick investigations without writing any code, directly in a web browser
- Deliver data to users:
  - Only events passing cuts and only variables user needs.
  - No need for experiment specific libraries - make it easier to use machine learning tools like Spark, Jupyter, R, SciPy, Caffe, TensorFlow. This would simplify Machine Learning Challenges: Higgs Boson ML Challenge, the Tracking challenge, ...
- Serve as a backend to:
  - Event Viewers (VP1, ATLANTIS, ATLASrift,...), make them platform independent
  - All kinds of education and outreach tools

The question is would the same technology map well on event data? Would event data structure map correctly, would data size be acceptable, what indexing and retrieval rate could we achieve?

Related work

EventIndex - a complete catalogue of all ATLAS events.

- Collection:ActiveMQ, storage:Hadoop, indexing:HBase, custom WebUI, CLI.
- ~1300 B/event. Event identifier, trigger pattern, hit counts, pointers to the event in all processing stages.
- Used for: event picking, event skimming, Panda event service
- Contains ~ $40*10^9$ data and $25*10^9$ MC event records. Searches at 10ms/event.

Open data - http://atlas-opendata.web.cern.ch/

- Provides access to real and simulated data, and educational tools, to the public
- Three levels of sophistication (simple visualizations, Jupyter based analysis, full set of analysis tools to download - documentation, datasets, software, VMs)

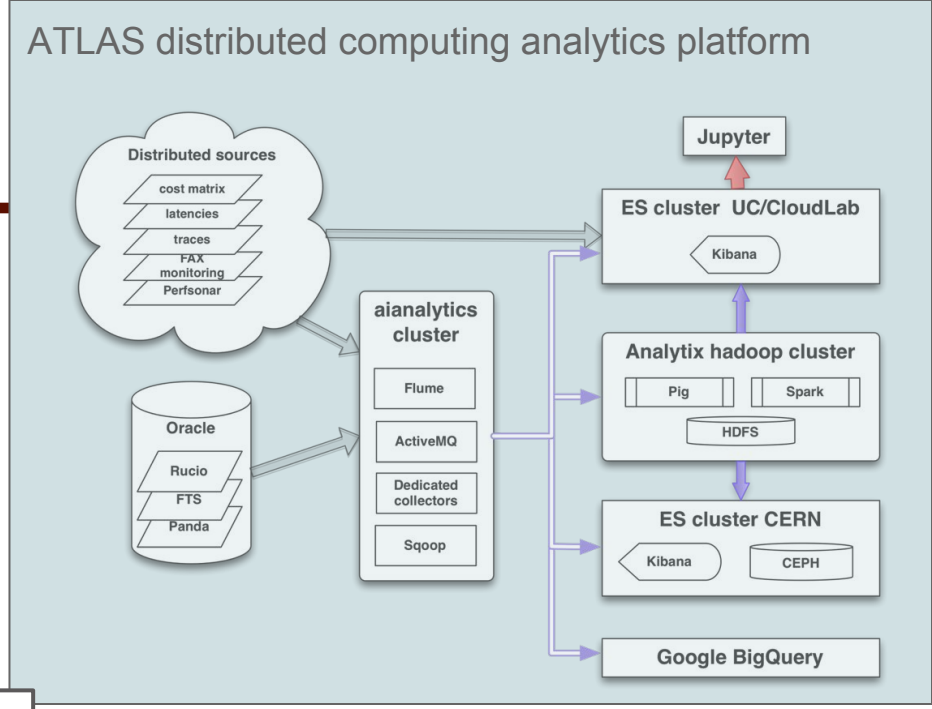
Lukas Heinrich's **project Aretha** https://www.youtube.com/watch?v=S5SkTXhdCng

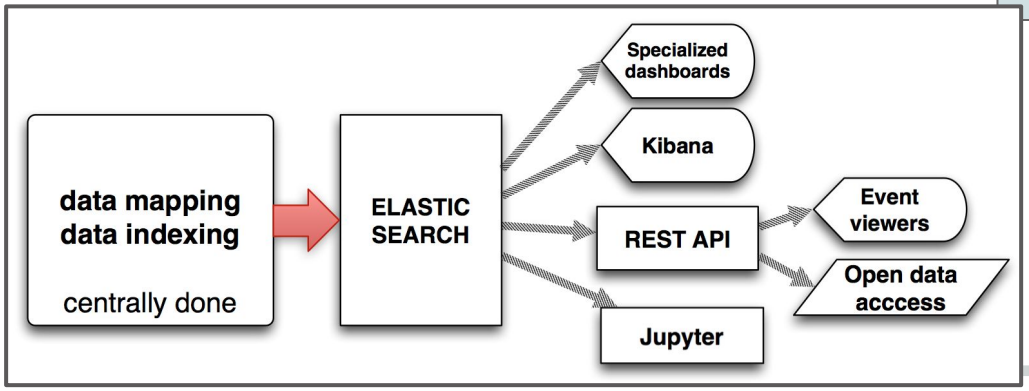
- Streams generated HepMC data from SHERPA, Herwig, MadGraph (locally or in cloud) to ZeroMQ
- Runs ZeroMQ data collector, Elasticsearch, Kibana all in docker
- Makes plots while jobs running, validate data using eg. Rivet

Analytics platform

There is a number of possible workflows that would work for storing and processing event data while using already existing ADC analytics platform:

- Store data in Hadoop (avro) process using pig or Spark. Not interactive, not a data access solution.
- Store in HDF5, use Spark. Performant analysis, does not solve data access.
- ...





Our selected workflow:

- Data indexed in Elasticsearch
- Fast visualization in Kibana
- Open data access through Elasticsearch REST API
- Data analysis on a co-located Jupyter cluster

Elasticsearch cluster

Elasticsearch is a distributed, real-time data and analytics, high availability open source search engine built on top of Apache Lucene. Works with structured JSON documents, schema free, by default all fields are indexed, data are compressed.

Indices:	Memory: 246GB /	Total Shards:	Unassigned Shards:	Documents:	Data:	Uptime: 4
2088	433GB	18644	0	8,481,799,362	9TB	months

THE UNIVERSITY OF CHICAGO

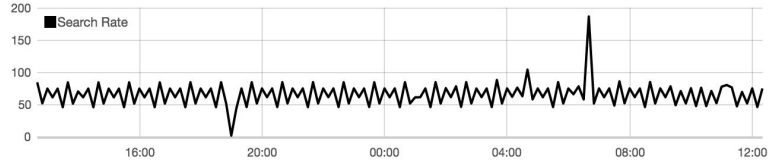
5 data + 3 head nodes

E5-2620v4, 64GB RAM, 4x800 GB SSD, 10Gbps NIC

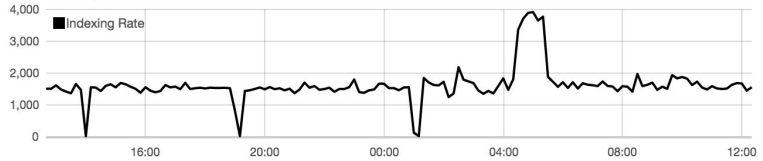
Used for ATLAS Distributed Computing analytics

Quite high base load

Search Rate: 74.46 /s



Indexing Rate: 1,536.95 /s



Jupyter

At University of Chicago

- Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz x2
- Tesla K20c x2
- 128GB RAM , 4.5TB RAID-5 for /scratch, 1TB RAID-1 for /
- If needed one more machine with 2 x Xeon Phi (64 core)
- Simple universal user, pass authentication, notebooks automatically checked in a git repo.

At CERN

- SWAN (Service for Web based Analysis)
- Currently in Beta
- Code in CERNBox

Advantages:

- Most of ML libraries installed, if there is anything missing we can easily add it.
- Local to Elasticsearch
- Learn from codes of others. See what was already done.

- Managed by CERN
- Your notebooks are yours only
- One can have the same environment anywhere you have CVMFS, just do:

```
setupATLAS
lsetup "lcgenv -p LCG_85swan3 x86_64-slc6-gcc49-opt ROOT rootpy ..."
```

ATLAS Data Formats

- Consider only user oriented DxAOD format.
- xAOD is a common centrally produced data format ~PB size.
- DxAODs are ~TB scale derivations, centrally produced, skimmed/slimed and customized for different physics groups use.
- Mini xAODs or flat n-tuples are subgroup specific and not considered here (~GB scale) .

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

Mapping events

We used root_numpy and rootpy.

We create document mapping template

- One per data format
- Describes event in terms of native Elasticsearch types and structures
- Tree is not very deep (5 levels) but needs care to get it right (array vs object vs nested object)

There is no need to index all of the variables, we'll still be able to deliver them, just not search/filter on them. Possible large improvement on data size (not used for this study).

There is a limit on 50 nested objects / document.

ROOT structure

```
*Branch  :EventInfoAux.
*Entries :     6752 : BranchElement (see below)
*.....*
*Br    2 :EventInfoAux.xAOD::AuxInfoBase : BASE
*Entries :     6752 : Total  Size=     529349 bytes  File Size  =      21915
*Baskets :       17 : Basket Size=     104448 bytes  Compression=  24.12
                                                                  ..*
                                                                    *
                                                  File Size  =      2124
                                                  Compression=  13.46
```

numpy objects

```
float32 AntiKt4EMTopoJetsAuxDyn Sp
[ 570.70159912  184.66441345]
float32 AntiKt4EMTopoJetsAuxDyn Split34
[ 210.48747253  159.41931152]
int32 AntiKt4EMTopoJetsAuxDyn ConstituentScale
[0 0]
object AntiKt4EMTopoJetsAuxDyn SumPtTrkPt1000
[ array([ 2215.11962891,     0.
1890.73132324,
        0.      ,    0.      ,    0.      ,
        0.      ,    0.      ,    0.      ,
ar
```

Mapping template

```
"EventInfoAuxDyn": {
  "type":"object",
  "properties":{
    "HLT_xe70_mht":{"type":"boolean"},
    "HLT_xe70_tc_lcw":{"type":"boolean"},
    "HLT_xe80":{"type":"boolean"},
    "HLT_xe80_mht":{"type":"boolean"},
    "HLT_xe80_tc_lcw":{"type":"boolean"},
    "mpx":{"type":"double"},
    "name":{"type":"string","index":"not_analyzed"},
  }
},
```

Indexed event

```
    "_score" : 1,
    "_source": {
      "EventInfoAuxDyn": {
        "HLT_xe70_tc_lcw": false
        "HLT_xe80": false,
        "HLT_xe80_tc_lcw": false
        "HLT_xe80_mht": false,
        "HLT_xe70_mht": false
      },
      "HLT_xAOD__JetContainer_a1
      "HLT_xAOD__JetContainer_a4
      "PrimaryVerticesAuxDyn": {
      "METAssoc_AntiKt4EMTopoAuxDyn": {
        "calsumpt": [
          [
            164343.22469163086,
            337029.10201851
          ],
```

Indexing events

We indexed di-jet events xAODs, and DxAODs from DAOD_EXOT2 derivation.

Used indexing method proved rather inefficient (parsing all the event structure for all events), also process was not tuned (batch size, etc).

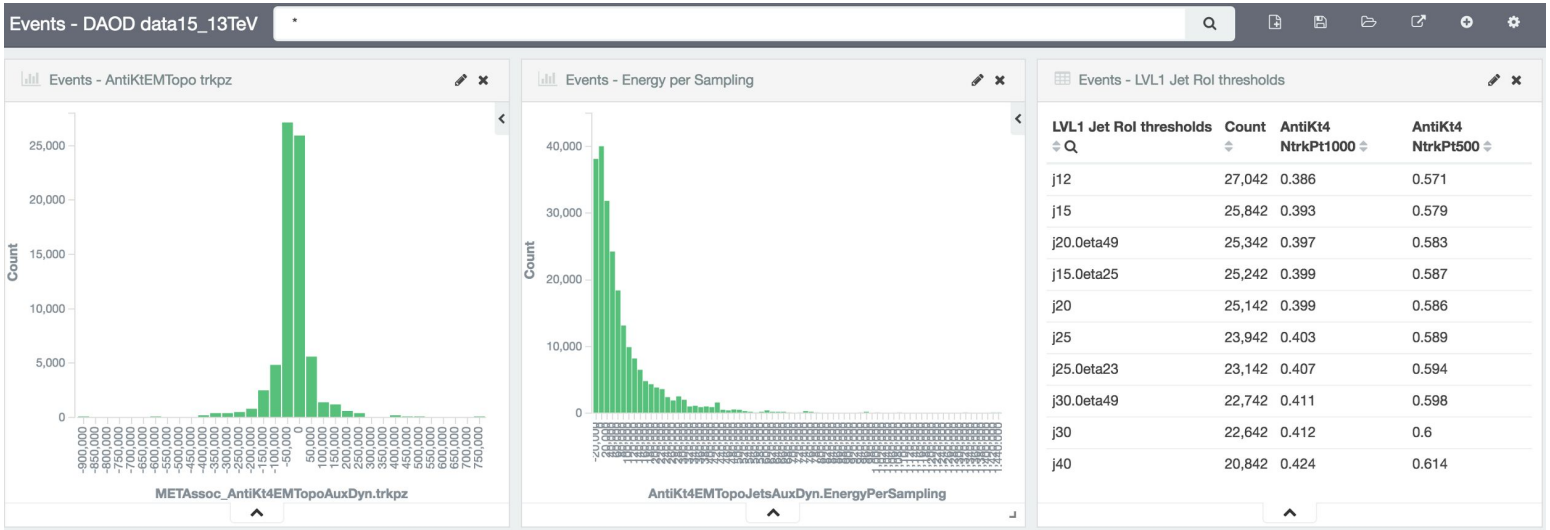
type	Event size		Indexing rate [Hz]
	[kb/event]	increase	
DAOD data15_13TeV	146	5.4	21
AOD data15_13TeV	377	2.0	8
DAOD mc15_13TeV	116	4.6	19
AOD mc15_13TeV	385	1.8	9

Data visualizations

Bad
Visualizations not made with scientists in mind.
Only simple operations supported.

Kibana can be used to do quick visualizations.
Use filter bar to make cuts.
Define additional derived variables.

Good
Visualizations can be organized and saved in nice reusable dashboards.
As all computation are distributed over all the ES nodes, visualizations are very fast even with billions of events.



LVL1 Jet RoI thresholds	Count	AntiKt4 NtrkPt1000	AntiKt4 NtrkPt500
j12	27,042	0.386	0.571
j15	25,842	0.393	0.579
j20.0eta49	25,342	0.397	0.583
j15.0eta25	25,242	0.399	0.587
j20	25,142	0.399	0.586
j25	23,942	0.403	0.589
j25.0eta23	23,142	0.407	0.594
j30.0eta49	22,742	0.411	0.598
j30	22,642	0.412	0.6
j40	20,842	0.424	0.614

Filtering and aggregations

Streaming full events:

```
"query" : {   "match_all": {} }
```

To return only some variables add:

```
"fields" : ["AntiKt4EMTopoJetsAuxDyn.ActiveArea", ...],
```

To select on a variable (cut):

```
"query" : { "term" : { "EventInfoAuxDyn.HLT_xe80" : True } }
```

Complex conditions (must, must_not, should, filter):

```
'bool':{
    'must':[
        {"term"  : { "EventInfoAuxDyn.HLT_xe80" : False }},
        {"range" : { "AntiKt4EMTopoJetsAuxDyn.GhostAntiKt2TrackJetCount" : { "gt": 2 } } },
        {"range" : { "AntiKt4LCTopoJetsAuxDyn.JetEMScaleMomentum_m": { "gt": 10000 } } },
        {"exists" :{ "field" : "HLT_xAOD__JetContainer_a4tcemsubjesFSAuxDyn.HECQuality"
}}
    ]}
```

Aggregations (cut and count analysis), very powerful, fast, can be nested. In our tests all are sub-second duration operations.

Simple count:

```
 "aggs": {   "docs": { "value_count": { "field": "_type" }  } }
```

Returning stats:

```
"aggs": {   "ptStats": { "stats": { "field": "TrackPt" }  } }
```

Stats in bins:

```
"aggs": {   "ptStats": { "histogram": {
          "field": "TrackPt",
          “Interval” : 50
} } }
```

Skimming & slimming performance

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 [%]
1 variable from all events	486	1	264	1	520	1	225	1
10 variables from all events	453	3	263	1	491	3	227	1
10 variables from events passing cut (~2 % events)	606 / 32 kHz	22	251 / 13 kHz	10	971 / 50 kHz	15	226 / 11 kHz	20
Full events passing cut (~2% events)	161 / 6.3 kHZ	77	35 / 1.8 kHz	84	121 / 6.2 kHz	80	32 / 1.5 kHz	82
Streaming all full events	116	86	39	87	120	88	31	86

Selections are basically free, cost is in data transfer back to client and client parsing JSON.

*client CPU usage.

Data modifications

Simple fixes can be done by:

- Adding corrected variable
- Defining “scripted” variables

Re-calibration from existing data would be performed by a central batch re-indexing.

Any kind of **reweighting** can be done by defining custom scoring function.

Weighting:

```
"script_score": {
    "script": {
        "lang": “lang”,
        “params”:  {
            "param1": value1,
            “param2” :value2
        } ,
        “Inline”:”_score * doc[‘orig_pt’].value /
pow(param1, param2)”
    }
}
```

Conclusions

- Feasibility of actual event indexing is tested
- Input/output rates are reasonable, increase in data volume is at the expected level.
- All kinds of optimizations are possible
  - Indexing directly from ATLAS analysis framework
  - Optimized mapping
  - Compressed responses
- Visualizations, skimming and sliming are very fast, very convenient for cut flow type analyses
- Full data retrieval while moderately slow, consumes too much CPU in parsing JSON returned.
- Elasticsearch is easy to scale up to a data center level (~ few hundreds of servers), bigger than that one need to add “tribe” nodes.