HEPDOOP: High-Energy Physics Analysis using Hadoop

W. Bhimji, T.Bristow, A.Washbrook

SUPA School of Physics and Astronomy, University of Edinburgh, Edinburgh, UK

E-mail: `wbhimji@staffmail.ed.ac.uk`

Abstract. We perform a LHC data analysis workflow using tools and data formats that are commonly used in the "Big Data" community outside High Energy Physics (HEP). These include Apache Avro for serialisation to binary files, Pig and Hadoop for mass data processing and Python Scikit-Learn for multi-variate analysis. Comparison is made with the same analysis performed with current HEP tools in ROOT.

1. Introduction

"Big Data" has now moved beyond buzzword to business-as-usual in the private sector including big names such as Yahoo, Twitter, Facebook, Amazon and Google. High-Energy Physics (HEP) is often cited as the archetypal Big Data use case, however it currently shares very little of the toolkit used in the private sector which is dominated by the Apache Hadoop ecosystem. We use those tools to perform aspects of a HEP analysis as part of a longer term goal of interaction between big data communities. For this study we use established tools with large user communities (see below). We aim for performance that is good-enough (i.e. of the same order as HEP tools rather than seeking to exceed it) and make ease-of-use also an important consideration.

1.1. Technologies used

Hadoop is a framework "for the distributed processing of large data sets across clusters of computers using simple programming models"[1]. It provides easy execution of MapReduce programs [2] and the HDFS distributed filesystem [3].

Pig produces MapReduce programs for Hadoop using a query language called Pig Latin [4] which offers [5] "ease of programming" for parallel execution of tasks, "optimisation opportunities" as task execution is optimised automatically, and "extensibility" through User Defined Functions (UDF)s.

Avro [6] is a file-based binary-format data serialization system. It relies on schemas but these are stored in the file which are thus self-describing. Files are compressed, with a choice of codecs, and tools for reading/writing are available in several languages including Pig. It is not a columnar store, though these formats are moving in that direction following Google's *Dremel* paper [7]. *Parquet* [8] is perhaps the best documented of the column-oriented products but was still considered too immature for the initial stage of this project.

Scikit-Learn [9] is a mature, well-documented and fully-featured machine learning package

for python built on NumPy and matplotlib. The later are also used for visualisation and histogramming in this project.
ROOT [10] [11] is the most commonly used data analysis technology in HEP both for data structures and the analysis toolkit. A number of tools have been developed for ROOT including **TMVA** [12][13] for multi-variate analysis.

2. HEP Analysis Use-case
An important workflow within HEP analysis is illustrated in figure 1. This illustation uses both terms employed in HEP but also more generally applicable titles for comparison with alternative disciplines. These stages are discussed in the sections that follow.

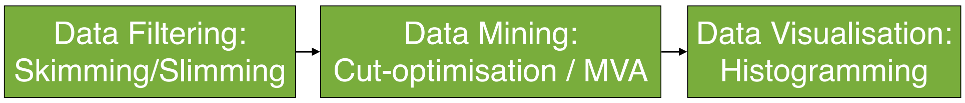
Data Filtering: Skimming/Slimming
Data Mining: Cut-optimisation / MVA
Data Visualisation: Histogramming

Figure 1. Illustrative HEP analysis workflow.

3. Data Filtering
We use data from the ATLAS experiment in the popular, centrally-produced, *SMWZ_D3PD* format which contains ROOT TTrees with 5860 branches of simple types and vectors with a total size of ≈65 KB/event. Due to the large file sizes, an analysis group will often conduct a "Filtering" stage to reduce the volume of data used in further analysis. The output is in the same structure but with a selection of events and reduction of branches. We use 1TB of input data which is converted into zlib-compressed Avro files using a simple conversion script based on the Avro python interface. We apply a selection motivated by an analysis of Higgs Boson decays to two b-quarks (H→bb) which has an event selection efficiency of 5% and selects 109 branches resulting in a file-size around 0.3% of the original. Illustrative Pig code is shown in figure 2. Filtering is done with simple comparisons (such as `MET_RefFinal_et` in this example) and more complex selections via a python UDF (`myfuncs.passesJetSelection` in this example). For the UDFs, the same code can be reused in both Pig and ROOT.

```
REGISTER 'MySelections.py' using jython as myfuncs;
DEFINE  AvroStorage org.apache.pig.piggybank.storage.avro.AvroStorage;
input = LOAD '$input' USING AvroStorage();
data = FOREACH input GENERATE $branches;
outputdata = FILTER data BY MET_RefFinal_et > 45.0
AND  myfuncs.passesJetSelection(jet_AntiKt4TopoEM)
AND ...
STORE outputdata INTO './results' USING AvroStorage();
```

Figure 2. Illustrative PIG code for *Data Filtering.*

3.1. Performance
The Hadoop processing time versus the size of the input dataset, when run on a 500 core Hadoop installation at CERN, is shown in figure 3. The ROOT-based tool centrally available on ATLAS for this purpose (filter-and-merge) has similar 270s run time on a single 760 MB file. However with custom ROOT code, reading only the branches used, we were able to achieve a single-file

run time of 30s and the application could be run in parallel on a cluster of this sort, with a batch system or using PROOF, possibly with better scaling than seen here for Hadoop. We do also observe, however, an improvement of performance in Hadoop/Avro when some large, unused, vector branches are removed to leave 2300 "branches" at 2.5KB/event.

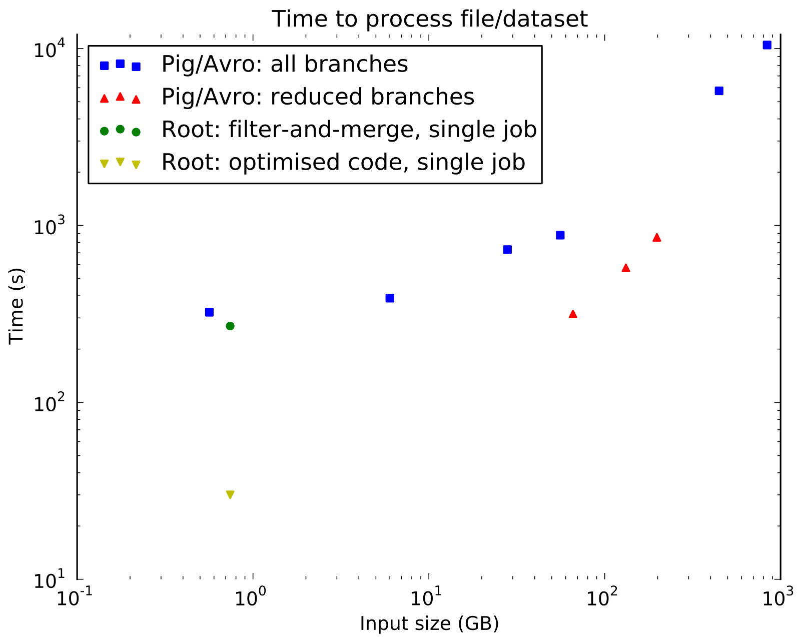

Figure 3. Time taken to perform filtering in Pig for files with all branches from the D3PD (blue squares) or when certain very large vector branches are removed (red triangles) together with that for ROOT running on a single 760 MB file using the filter-and-merge tool (green circle) or custom ROOT code reading only the branches of interest (yellow triangle).

4. Data Mining

Filtered data is then mined by applying further linear selections, which we implement in Pig, or a Multi-Variate Analysis (MVA) such as a Boosted Decision Tree (BDT), which we implement in scikit-learn. This is run on the complete input data for the ATLAS H→bb analysis. Illustrative code is shown in figure 4, In that code, *trainingData, trainingClasses* and *testingData* are NumPy arrays that can either be read from ROOT TTrees (using rootpy [14]) or filled from Avro using the python interface.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=4,compute_importances=True,
 min_samples_split=2,min_samples_leaf=100),n_estimators=400, learning_rate=0.5,
 algorithm="SAMME",compute_importances=True)
ada.fit(trainingData, trainingClasses)
return ada.decision_function(testingData)
```

Figure 4. Illustrative python MVA code

We use IPython[15] to enable parallel training on samples and achieve an execution time in Scikit similar to that of TMVA: for the training of 4 BDTs Scikit takes a combined time of around 180 $\pm$ 5 s while TMVA takes around 200 $\pm$ 5 s. Outputs of the BDT are shown in figure 5. We can recreate the same results for the analysis as obtained with ROOT/TMVA, but the strength of this package is that it also allows for a wide range of alternative approaches.

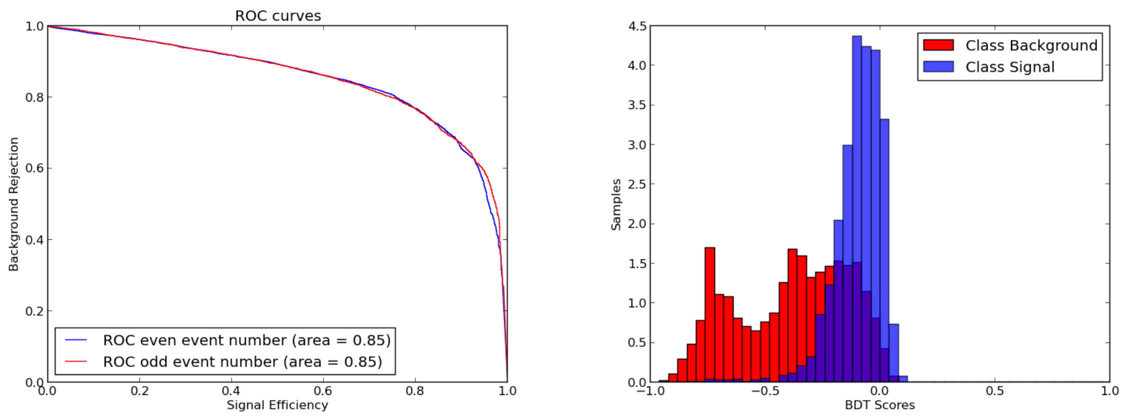

Figure 5. Outputs from the scikit-learn MVA

5. Conclusions

Key components of a real HEP analysis have been run using out-of-the-box Big Data tools in Hadoop and python. On advantage of this approach is that it can enable use of wider resources such as Amazon Elastic Map Reduce as well as benefitting from large user and developer communities and further off-the-shelf tools for use in, for example, provisioning, monitoring and scheduling. The performance seen here for filtering is, however, significantly inferior to our ROOT-based analysis, particularly in the case of large complex nested structures. To attempt to address this issue, we are exploring the use of the columnar storage format *Parquet* [8].

Acknowledgments

Thanks to the CERN IT-DSS group for use of Hadoop resources.

References

[1] URL http://hadoop.apache.org/ (Accessed October 2013)
[2] Dean, J. and Ghemawat, S. 2008 MapReduce: simplified data processing on large clusters *Communications of the ACM* **51(1)** 107-113
[3] Shvachko, K. *et al.* 2010 The hadoop distributed file system *Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* 1-10
[4] Olston, C., *et al.* 2008 Pig latin: a not-so-foreign language for data processing *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* 1099-1110
[5] URL http://pig.apache.org/ (Accessed October 2013)
[6] URL http://avro.apache.org/ (Accessed October 2013)
[7] Melnik, S. *et al.* 2010 Dremel: interactive analysis of web-scale datasets *Proceedings of the VLDB Endowment* **3(1-2)** 330-339
[8] URL http://parquet.io/ (Accessed October 2013)
[9] URL http://scikit-learn.org/ (Accessed October 2013)
[10] Brun R and Rademakers F 1997 *Nucl. Instrum. Meth.* A **389** 81
[11] URL http://root.cern.ch/ (Accessed October 2013)
[12] Hoecker, A. *et al.* 2007 TMVA - Toolkit for Multivariate Data Analysis arXiv:physics/0703039

[13] URL http://tmva.sourceforge.net/ (Accessed October 2013)
[14] URL http://www.rootpy.org/ (Accessed October 2013)
[15] Perez, F., and Granger, B. E. 2007 IPython: a system for interactive scientific computing *Computing in Science and Engineering* **9(3)** 21-29