Data Formats in HEP Analyses

Jakob Blomer

HEP Analysis Ecosystem Workshop, Amsterdam
May 22th, 2017

“HEP Analysis” Assumptions in this Talk

- Focus on the “last mile” of publication creation

- Data sets are relatively small
  - $\mathcal{O}(10^7)$ events
  - Tens to hundreds of properties per event
  - Volume: gigabytes to terabytes
- Processing tends to escape central computing workflows and resources
- Number of reads $>>$ number of writes
- I/O bound: little calculation

Little data lock-in:
final data sets can (sometimes need to be) reproduced
e. g. starting from CMS MINIAOD, ATLAS xAOD

What we want from the data format

1. Fast turn-around
   "Let me quickly plot. . . "
   - Tuning cuts (partial reading of the data set)
   - Feeding into ML framework (full reading of the data set)
2. Integration, unleashing the data
   - Libraries for C++, Python
   - Machine learning frameworks (e. g. TMVA, TensorFlow)
   - Big Data schedulers (e. g. Spark)
   - Analytic tool kits (e. g. ROOT, R, SciPy)

What we do not necessarily need:

- Plethora of tuning options (likely to be unused)

Storage Technology Variety

Universally efficient data layout and encoding is challenging
given large spectrum of storage performance characterisics

- Latency:
  10 ns RAM → 100 µs 3D XPoint → 1 ms SSD → 10 ms HDD, LAN
  between fastest and slowest: ×100 000
- Throughput:
  20 GB/s RAM → 2 GB/s SSD, 3D XPoint → 100 MB/s HDD, LAN
  between fastest and slowest: ×200

Additionally: uneven scaling of components, e. g.
memory size vs. network bandwidth in laptops

Functional Data Format Expectations

- Well-defined encoding, not simply a memory dump
- Self-describing, schema included
- Support for complex, nested data types (records, arrays)
- Preserving floating point precision
- Checksumming
- Possibly schema evolution

A Menu of Data Formats

ROOT:
- Stream of serialized C++ objects in columnar layout

Protobuf:
- Not a file format per se but (de)-serialization of small records
- Possible file format: `schema` || `[size record-blob]*`

SQlite:
- SQL database in a file

HDF5:
- Popular in the HPC universe
- There are HEP machine learning data sets in HDF5
- Many options, somewhat a file format toolkit
- Hierarchical "data sets" that each contain a "data space" (*n*-dimensional array)

Avro:
- From the Apache Hadoop/Spark universe
- JSON schema, row-wise binary storage

Parquet:
- From the Apache Hadoop/Spark universe
- *column*-wise binary storage

"Fruit Fly" Data Set: The LHCb OpenData Sample

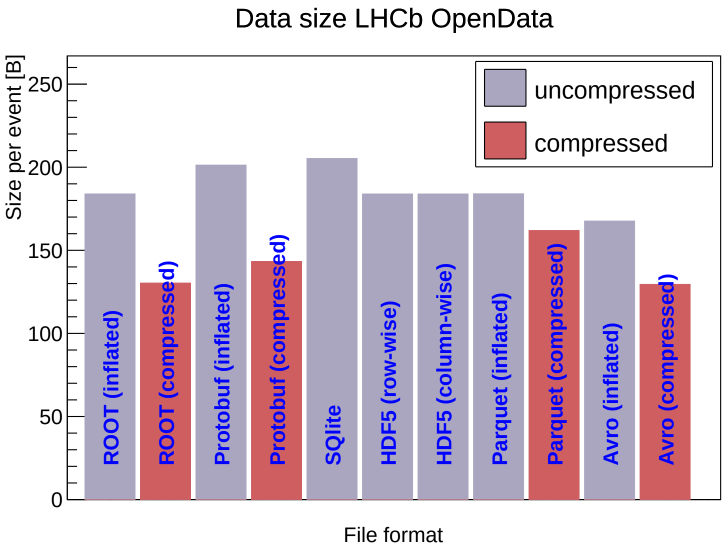
Starting point: "What if I had my data set in format X?"

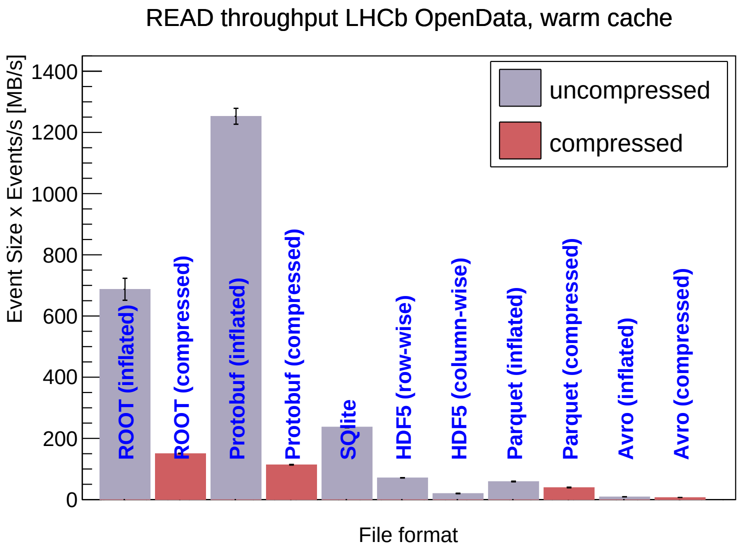
- 8.5 million run 1 events $B \to KKK$
- Flat n-tuple, 26 branches (mostly floating point numbers)
- 21 branches needed for the toy analysis
- 2.4 million events can be skipped
  because one of the kaon candidates is flagged as a muon
- Single-threaded

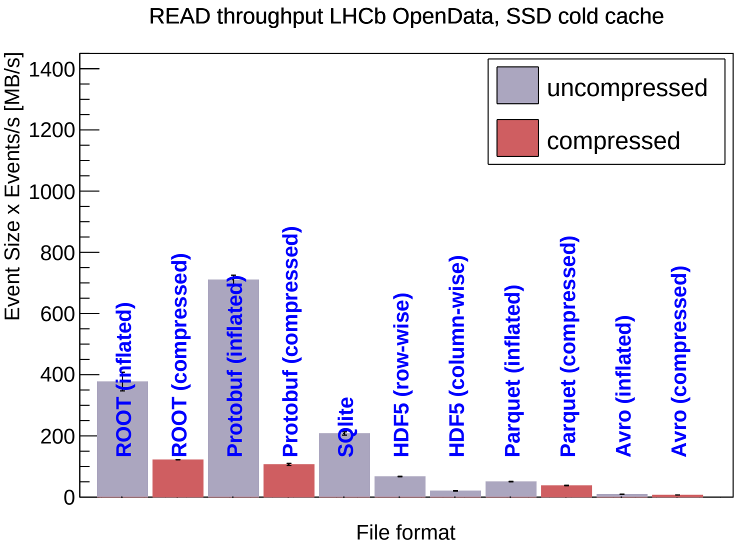
On the simple end of the spectrum,
helps to understand performance base case

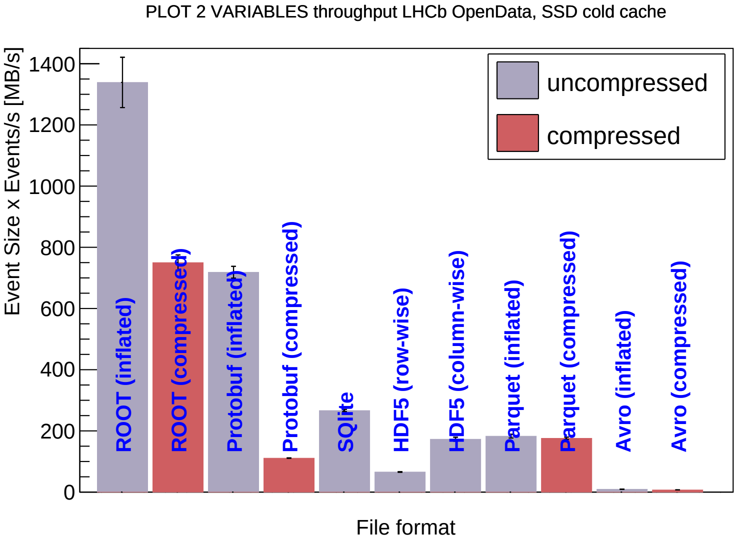
Links

https://github.com/lhcb/opendata-project
https://github.com/lhcb/opendata-project/blob/master/Background-Information-Notebooks/EventData.ipynb
https://github.com/jblomer/iotools/tree/hsf-analysis-workshop

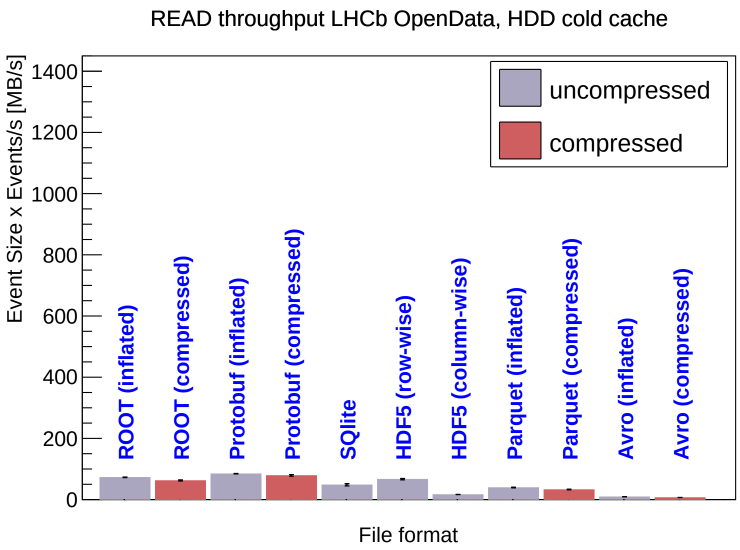

Data size LHCb OpenData
Size per event [B]
0
50
100
150
200
250
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format


READ throughput LHCb OpenData, warm cache
Event Size x Events/s [MB/s]
1400
1200
1000
800
600
400
200
0
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format

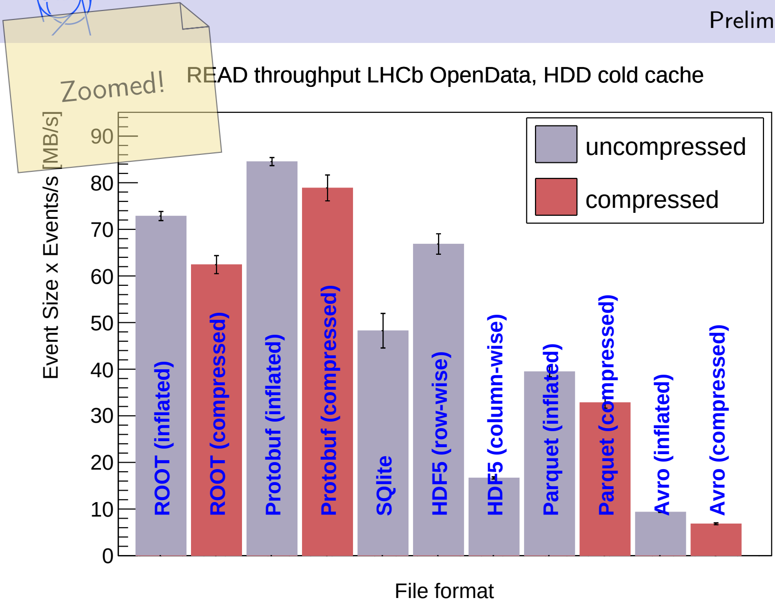

READ throughput LHCb OpenData, SSD cold cache
Event Size x Events/s [MB/s]
1400
1200
1000
800
600
400
200
0
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format

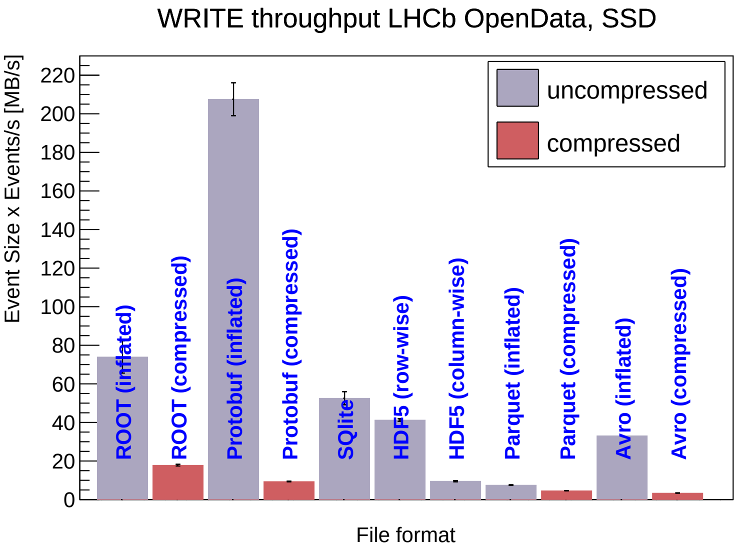


PLOT 2 VARIABLES throughput LHCb OpenData, SSD cold cache
Event Size x Events/s [MB/s]
1400
1200
1000
800
600
400
200
0
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format


READ throughput LHCb OpenData, HDD cold cache
Event Size x Events/s [MB/s]
1400
1200
1000
800
600
400
200
0
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format

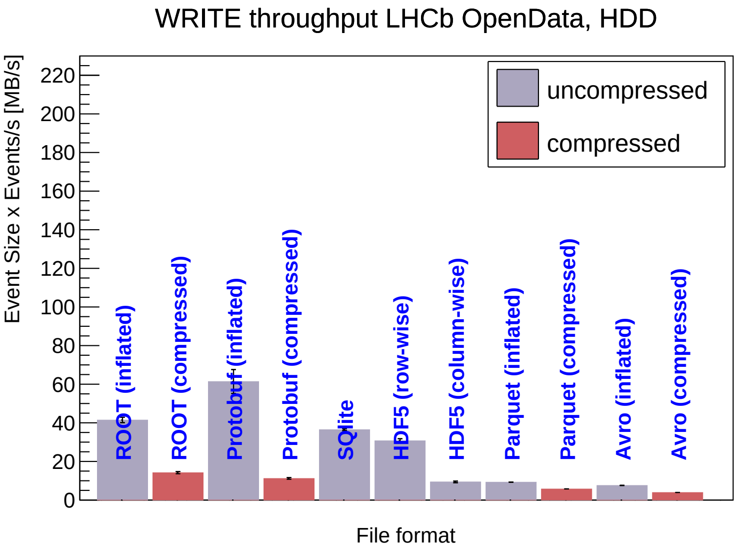
Read Throughput (Spinning Disk, Zoomed)




WRITE throughput LHCb OpenData, SSD
Event Size x Events/s [MB/s]
0
20
40
60
80
100
120
140
160
180
200
220
uncompressed
compressed
ROOT (inflated)
ROOT (compressed)
Protobuf (inflated)
Protobuf (compressed)
SQlite
HDF5 (row-wise)
HDF5 (column-wise)
Parquet (inflated)
Parquet (compressed)
Avro (inflated)
Avro (compressed)
File format

Write Throughput (Spinning Disk)

Preliminary Figures



Summary

This is an ongoing study. I'm happy to add more file formats and data sets.

Thoughts on the numbers:

- On fast storage, performance is dominated by (de-)serialization
- We might want to change the compression default on fast storage (or aim for hardware acceleration for compression algorithms)

A repository of **archetype analyses** would be very helpful.
Prepared data sets (not necessarily real data) *and*
a description of the analysis routine.

We can benefit a great lot from **integration work**.
That can mean teaching ROOT other data formats *or*
teaching 3rd party software the ROOT format.

Backup

Hardware	Type
CPU	i7-6820HQ @ 2.7 GHz
Memory	2×16 GB DDR4 2133 MHz
SSD (flash)	1 TB Toshiba XG3 PCIe
HDD (spinning)	Western Digital WD20NMVW

Library	Version
ROOT	6.08/06
protobuf	3.2.1
sqlite	3.18.0
hdf5	1.10.0_patch1
avro-c	1.8.1
parquet-cpp	1.0.0

Operating system: Linux 4.10, glibc 2.25, gcc 6.3.1
Compression refers to zlib (DEFLATE) with default compression level.