ROOT I/O compression algorithms and their performance impact within Run 3

Oksana Shadura, Brian Paul Bockelman
University of Nebraska–Lincoln, USA
oksana.shadura@cern.ch, bbockelm@cse.unl.edu

1. Introduction

We have performed a survey of the performance of the new compression techniques. We also provide insight into solutions applied to the bottlenecks in compression algorithms for the improved ROOT performance.

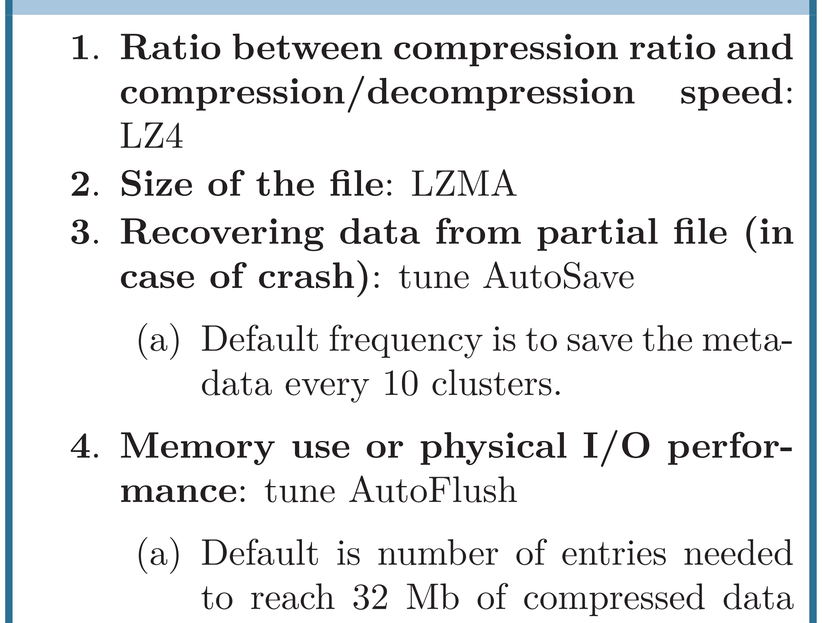
2. ROOT compression algorithms

1. **ZLIB** - a LZ77 preprocessor with Huffman coding [ROOT default]
   (a) other modifications - **zlib-cf** or **Intel zlib**
2. **LZMA** - a variant of LZ77 with huge dictionary sizes and special support for repeatedly used match distances, whose output is then encoded with a range encoder, using a complex model to make a probability prediction of each bit.
3. **LZ4** - a LZ77-type compressor with a fixed, byte-oriented encoding and no Huffman coding pass [new ROOT default]
4. **ZSTD** - a dictionary-type algorithm (LZ77) with large search window and fast implementations of entropy coding stage, using either very fast Finite State Entropy (tANS) or Huffman coding. [Facebook]
5. **Snappy** - a byte aligned LZ77 algorithm intended for high speed rather than good compression.[Google]
6. Old ROOT compression algorithm (backward compatibility)

5. Recommendation to the users

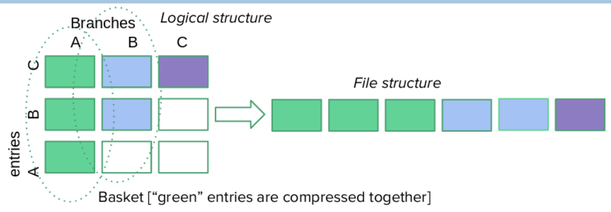
1. **Ratio between compression ratio and compression/decompression speed**: LZ4
2. **Size of the file**: LZMA
3. **Recovering data from partial file (in case of crash)**: tune AutoSave
   (a) Default frequency is to save the metadata every 10 clusters.
4. **Memory use or physical I/O performance**: tune AutoFlush
   (a) Default is number of entries needed to reach 32 Mb of compressed data [number of entries or compressed data size]

7. LZ4 + Bitshuffle preconditioner



→ Promising results that allow to outperform ZLIB compression ratio.

3. ROOT I/O overview



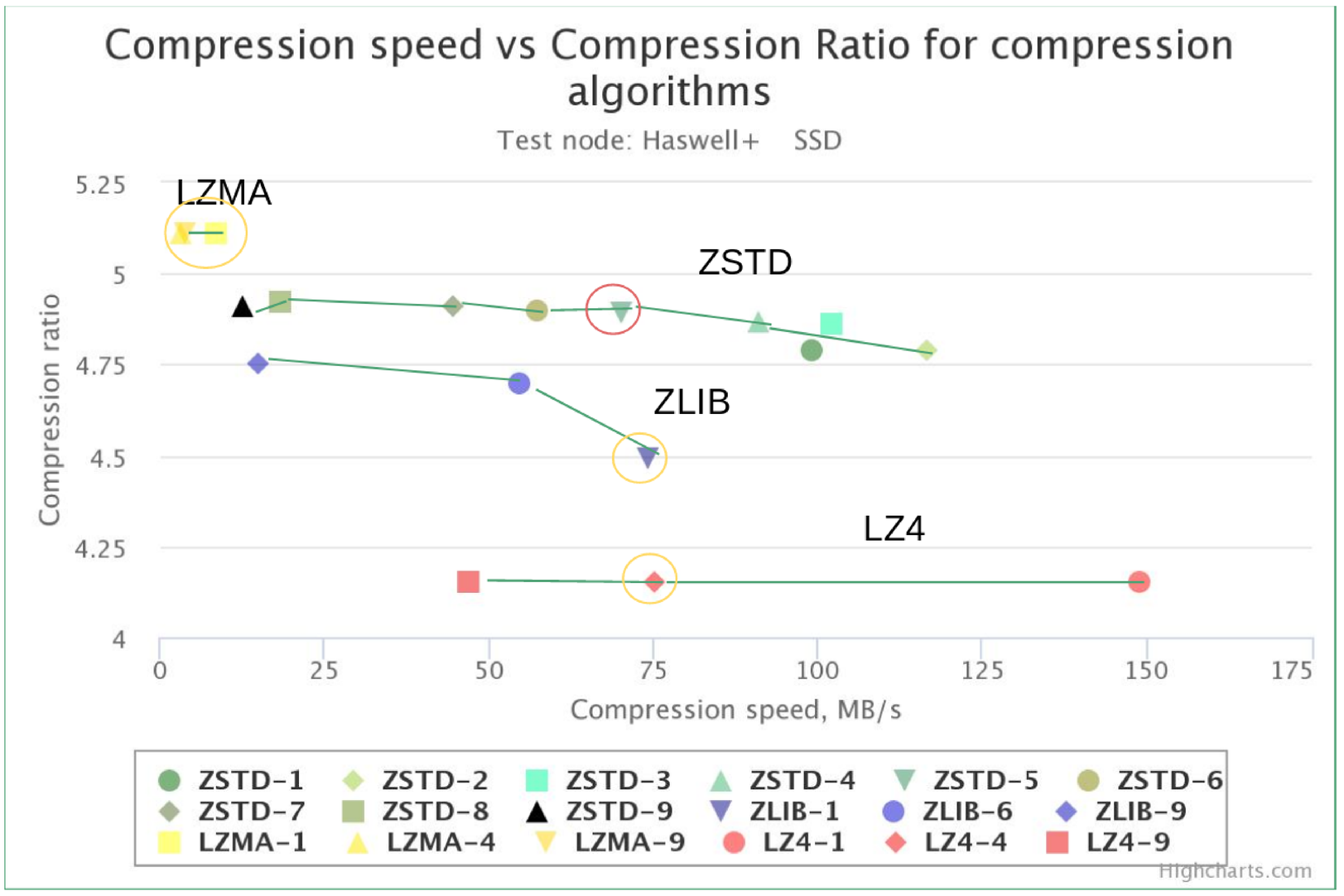
→ We can try to arrange bits of the values to efficiently compress sequences for primitive branches [e.g. Kudu, Apache].

4. Compression Pre-conditioners

Algorithms that rearranges typed, binary data for improving compression:

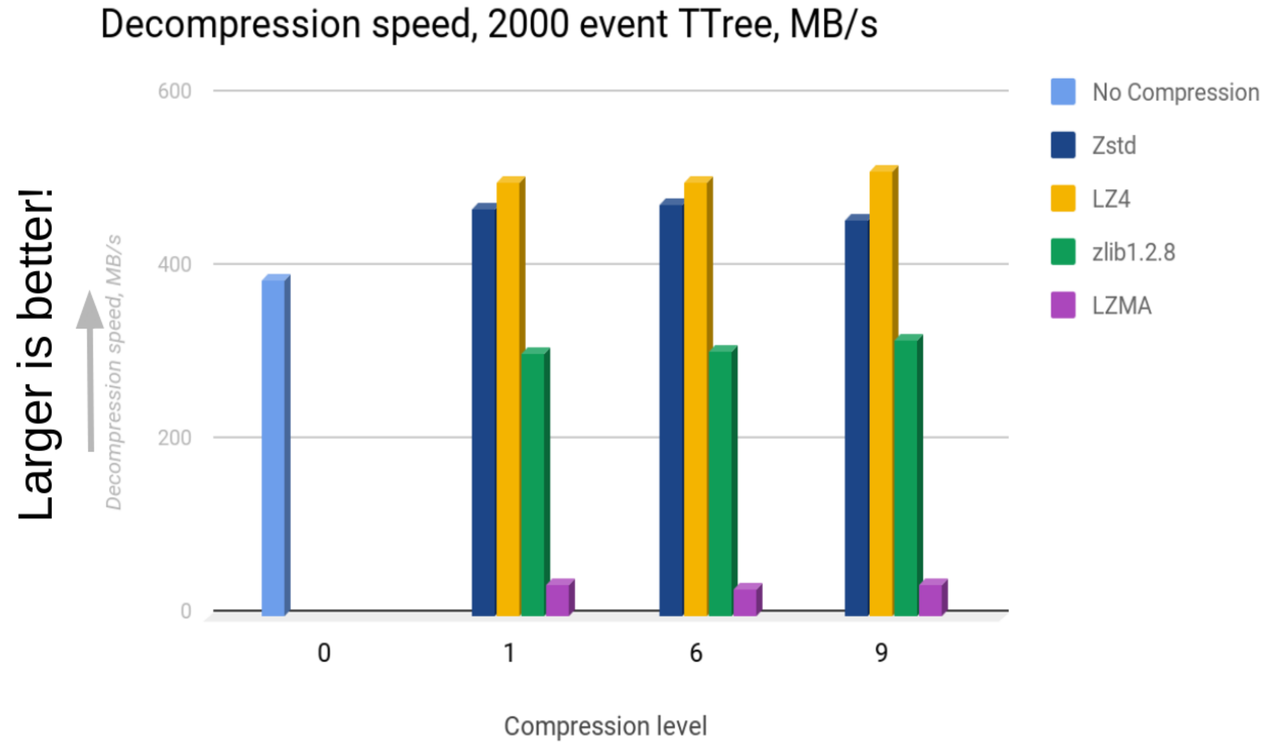
1. **Shuffle** - integrated byte shuffle preconditioner (available in Blosc)
2. **BitShuffle** - integrated bit shuffle preconditioner (available in Blosc)

6.Comparison of ROOT compression algorithms -compression



→ ZSTD is showing promising results, even though compression dictionary procedure is still not optimised.

9. Comparison of current ROOT compression algorithms - decompression



→ We know that reading with decompression for LZ4 and ZSTD is actually faster than reading decompressed: significantly less data is coming from the I/O subsystem.