Compression Update: ZSTD & ZLIB

Oksana Shadura,
Brian Bockelman
University of Lincoln Nebraska

Background: Compression algorithms comparisons

- As part of the DIANA/HEP to improve ROOT-based analysis, we have continued work in comparing compression algorithms. For this update, we include:
    - **ZSTD**: Relatively new algorithm in the LZ77 family, notable for its highly performant reference implementation and versatility.
    - **ZLIB / Cloudflare**: Update on work to include Cloudflare patches in ROOT.
- We will be comparing algorithms based on three metrics:
    - **Compression ratio:** The original size (numerator) compared with the compressed size (denominator), measured in unitless data as a size ratio of 1.0 or greater.
    - **Compression speed:** How quickly we can make the data smaller, measured in MB/s of input data consumed.
    - **Decompression speed:** How quickly we can reconstruct the original data from the compressed data, measured in MB/s for the rate at which data is produced from compressed data.

Testing setup - Software

- Performance numbers based on modified ROOT test “Roottest-io-compression-make” with 2000 events (unless noted).
- Branches:
  - https://github.com/oshadura/root/tree/latest-zlib-cms-cloudflare (latest cloudflare zlib, ported into ROOT Core)
  - https://github.com/oshadura/root/tree/brian-zstd (B.Bockelman’s ZSTD integration with CMake improvements)
  - https://github.com/oshadura/root/tree/zstd-default (branch enabling ZSTD as default, used only for testing purposes)
  - https://github.com/oshadura/roottest/tree/zstd-allcompressionlevels (roottest compression test with extended cases presented here, covering all zlib and zstd compression level)
- We are trying to measuring the ROOT-level performance - numbers include all overheads (serialization / deserialization, ROOT library calls, etc).

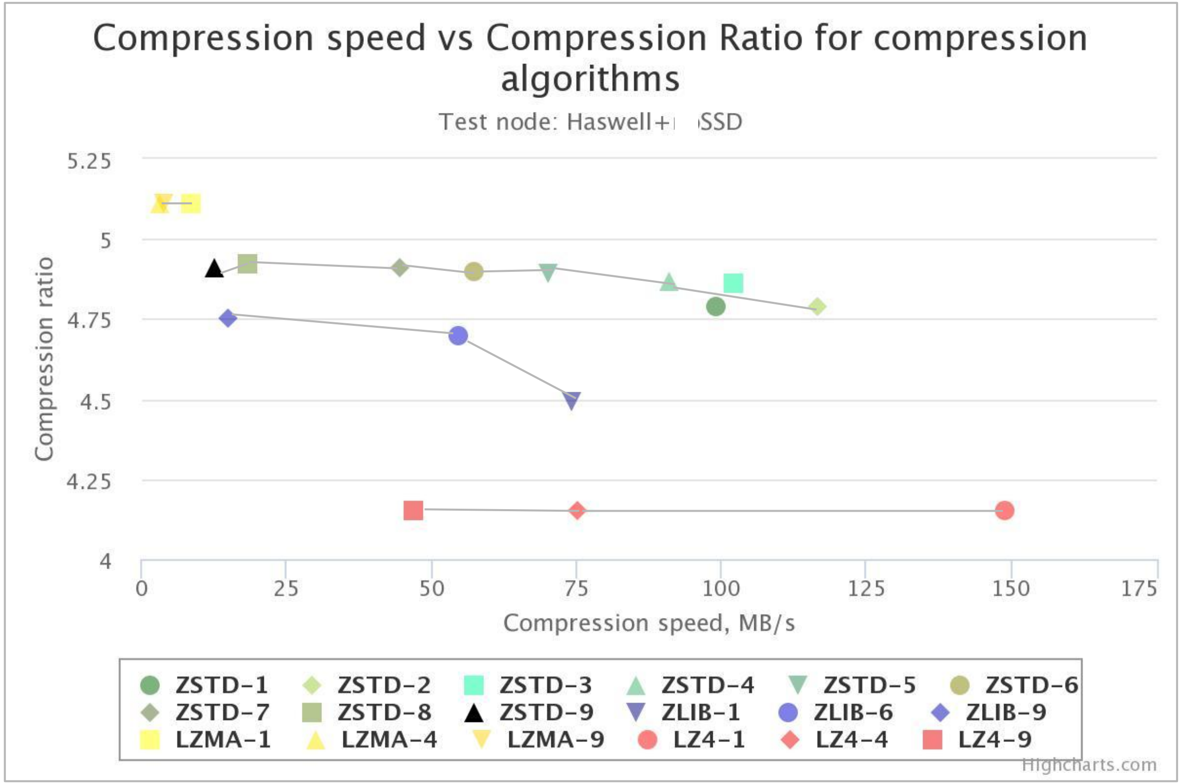
Testing setup - Hardware

- Platforms utilized:
    - **Intel Laptop**: Intel Haswell Core i7 + SSD
    - **Intel Server**: Intel Haswell Xeon-E5-2683
    - **AARCH64neon Server:** Aarch64 ThunderX
    - **AARCH64neon+crc32 Server**: Aarch64 HiSilicon's Hi1612 processor (Taishan 2180). Includes CRC32 intrinsic instruction.
- Tests were repeated multiple times to give a sense of performance variability.

ZSTD Background

- Given ZSTD performance claims on their website (facebook.github.io/zstd/), we should expect:
    - **Better than ZLIB in all metrics**: compression speed, decompression speed, and compression ratio.
    - Like all LZ77 variants, **decompression speed should be constant regardless of compression level**.
    - High dynamic range in tradeoff between compression speed and compression ratio.
    - Does not achieve compression ratio of LZMA.
    - Does not achieve decompression speed of LZ4.

Write Tests - Write Speed and Compression Ratio



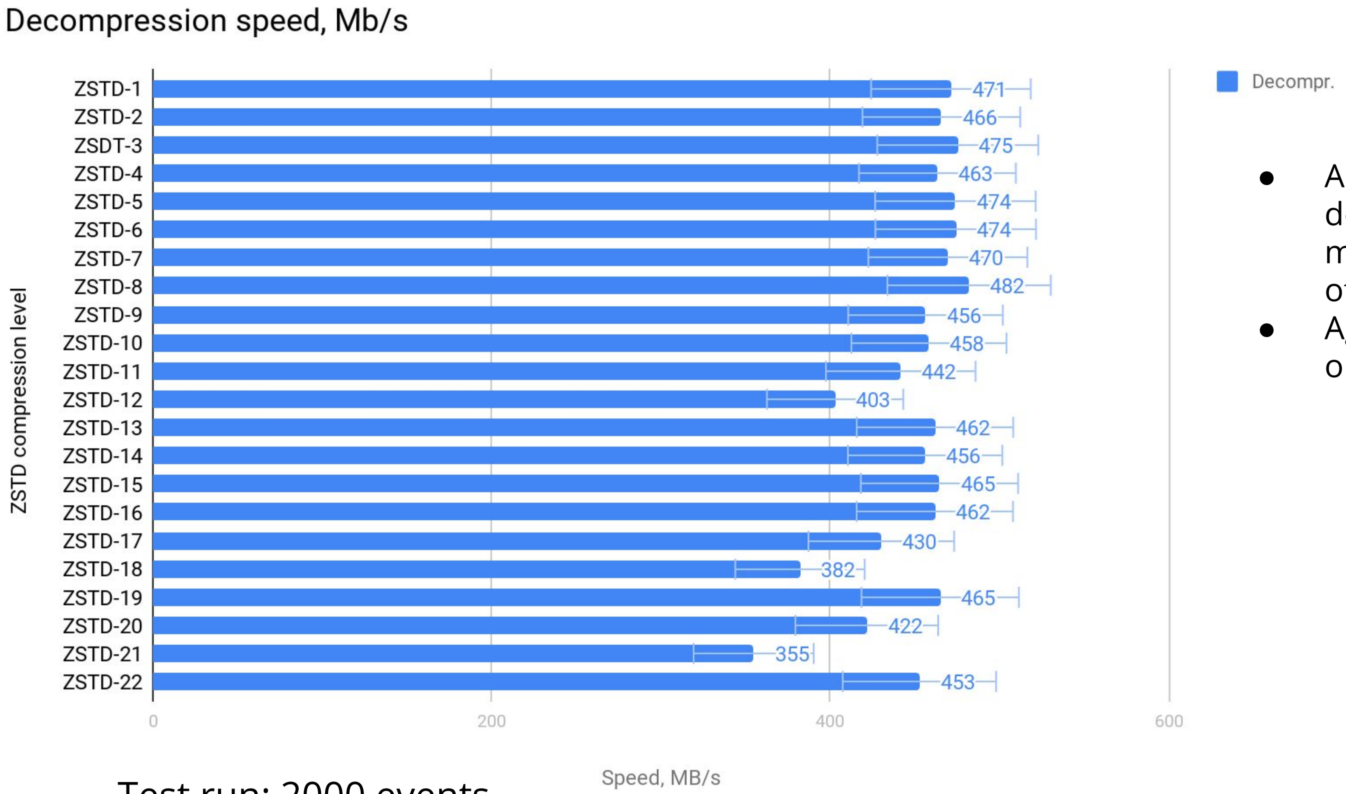
Larger is better

Larger is better

- Largely validates our expectations for compression!
- Note there is some performance noise between ZSTD-1 and ZSTD-2. Not understood.
- **NOTE**: Compression ratios are flatter than expected. Will do cross-comparisons with LHC files in a future follow-up.

Test used: roottest-io-compression-make with 2000 events
Raw data: http://jsfiddle.net/oshadura/yzusyhco/show/

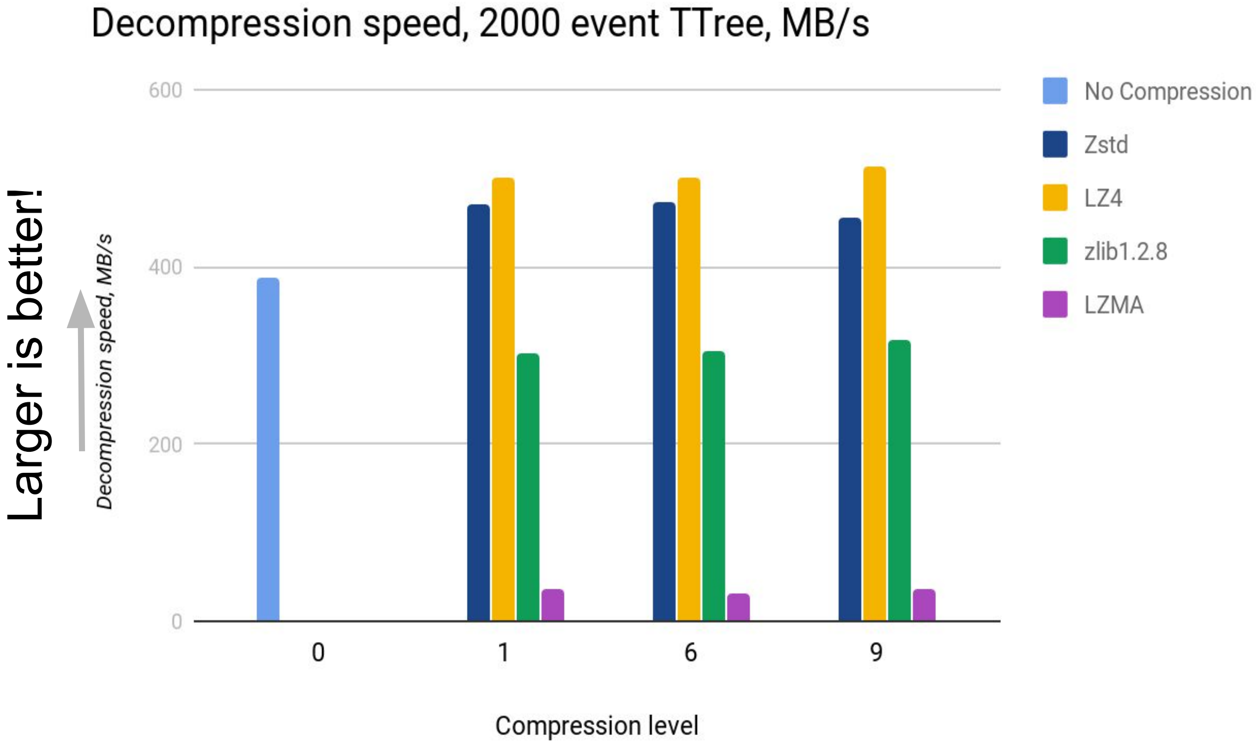
ZSTD - Read Speed Tests (Intel Laptop)



- As expected, decompression rates are mostly identical, regardless of compression level.
- Again, some curious outliers.

Test run: 2000 events
TTree-roottest-io-compression-make

Read Speed - Compare across algorithms



Larger is better!

- At the current compression ratios, reading with decompression for LZ4 and ZSTD is actually faster than reading decompressed: significantly less data is coming from the IO subsystem.
- We know LZ4 is significantly faster than ZSTD on standalone benchmarks: likely bottleneck is ROOT IO API.

ZSTD - Next steps:

- Follow-up with a wider corpus of inputs (e.g., LHCb ntuples, CMS NANOAOD).
- These tests indicate ZSTD would be a versatile addition to ROOT compression formats.
- Worthwhile to explore read rates for LZ4-vs-ZSTD: can we show cases where reading LZ4 is more significantly faster?
- ZSTD has an additional promising mode where the compression dictionary can be reused between baskets.
  - Facebook reports dictionary reuse provides massive improvements over baseline ZSTD for compression / decompression speeds and compression ratio when compressing small buffers (ROOT's use case!).
  - Naive tests did not bear out this claim: however, Facebook tested against a text-based corpus while we have binary data.
  - Needs investigation.

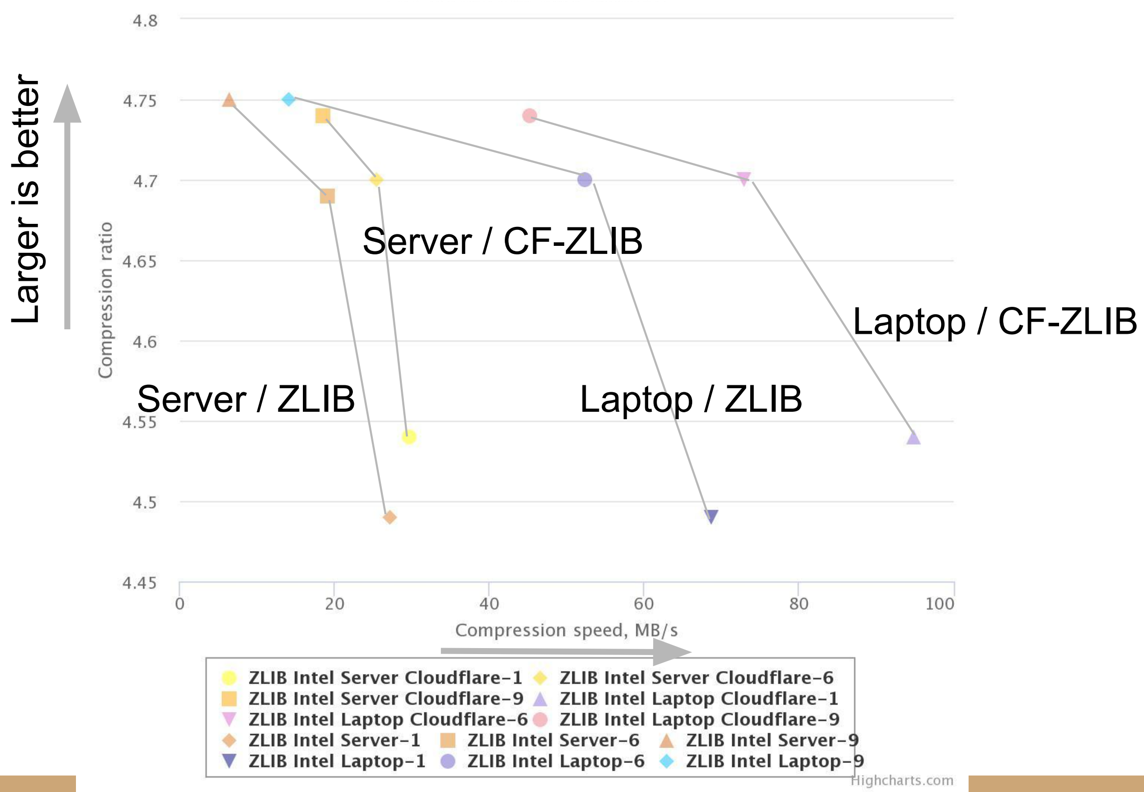
ZLIB Progress

- We have been trying to land the Cloudflare ZLIB (“CF-ZLIB”) patches into ROOT.
- ZLIB current version is 1.2.11; CF-ZLIB is based on 1.2.8.
    - Difference between 1.2.11 and 1.2.8 are mostly for build systems, bug fixes, and regression fixes in parts of the library unrelated to ROOT.
    - Rebasing Cloudflare to 1.2.11 proved very difficult. Decided to stay on 1.2.8.
- In addition to CloudFlare patches, we have added:
    - “Fat library”: When intrinsics are not available at runtime, switch to base implementation.
    - Build improvements: Now builds on ARM and Windows.
    - adler32 optimization: CloudFlare only optimizes CRC32; ROOT uses adler32.
- Here, we compare CF-ZLIB with upstream ZLIB.

Cloudflare ZLIB vs ZLIB - Intel Laptop/Intel Server (http://jsfiddle.net/oshadura/npp670kr/show)

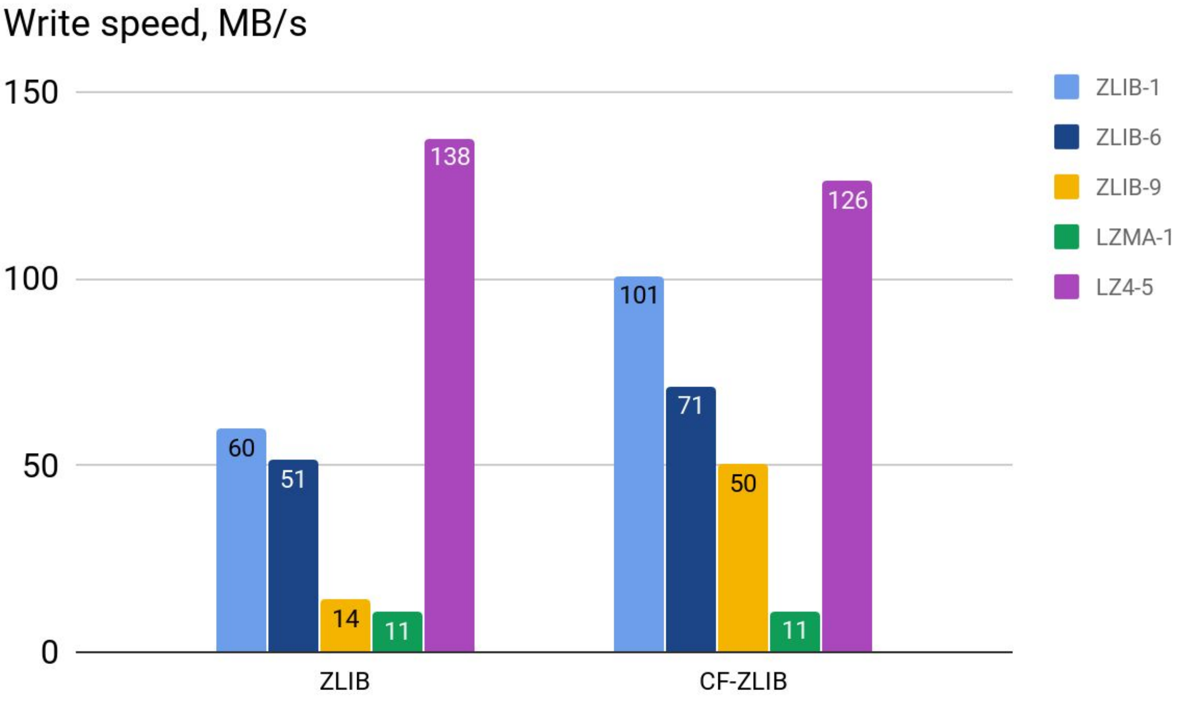
Note: small dynamic range for y-axis.

The CF-ZLIB compression ratios *do* change because CF-ZLIB uses a different, faster hash function.



Compression speed vs Compression Ratio for compression algorithms

Compression write speed (Intel Laptop)

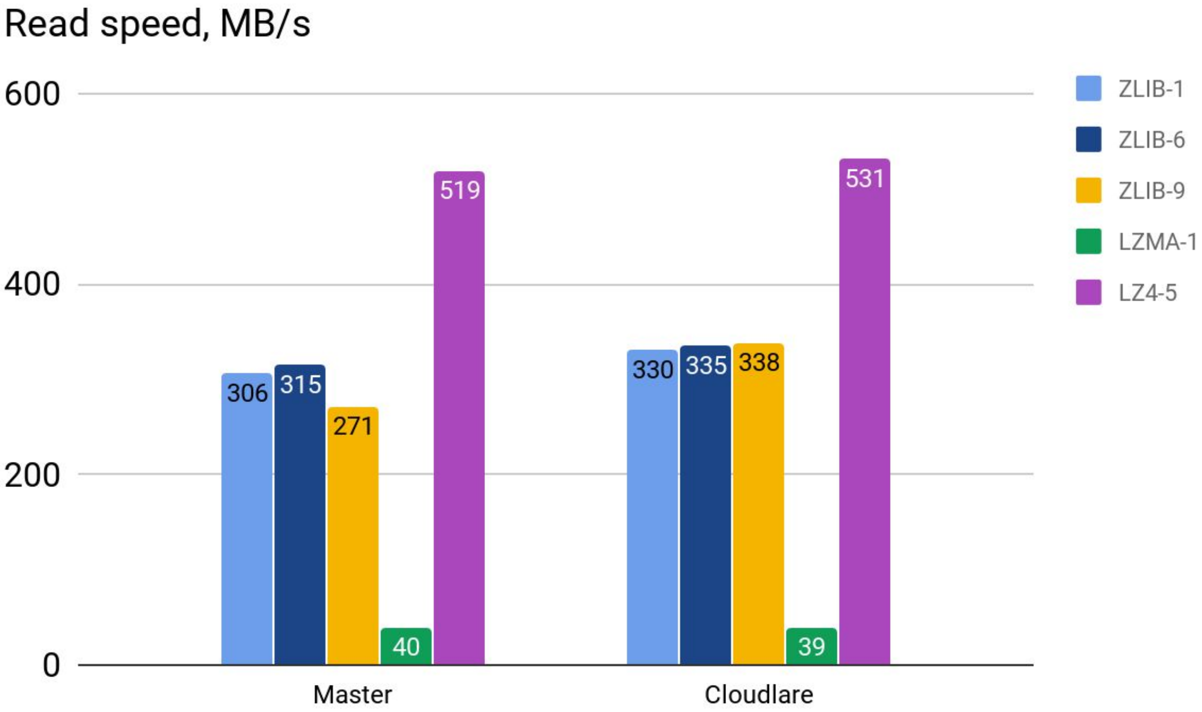


Reductions in speed:

- ZLIB-1: -40%
- ZLIB-6: -28%
- ZLIB-9 -72%

CF-ZLIB-9 is the same speed as ZLIB-6.

Read speed (Intel Laptop)



Small improvement of CloudFlare's version ~ 7%.

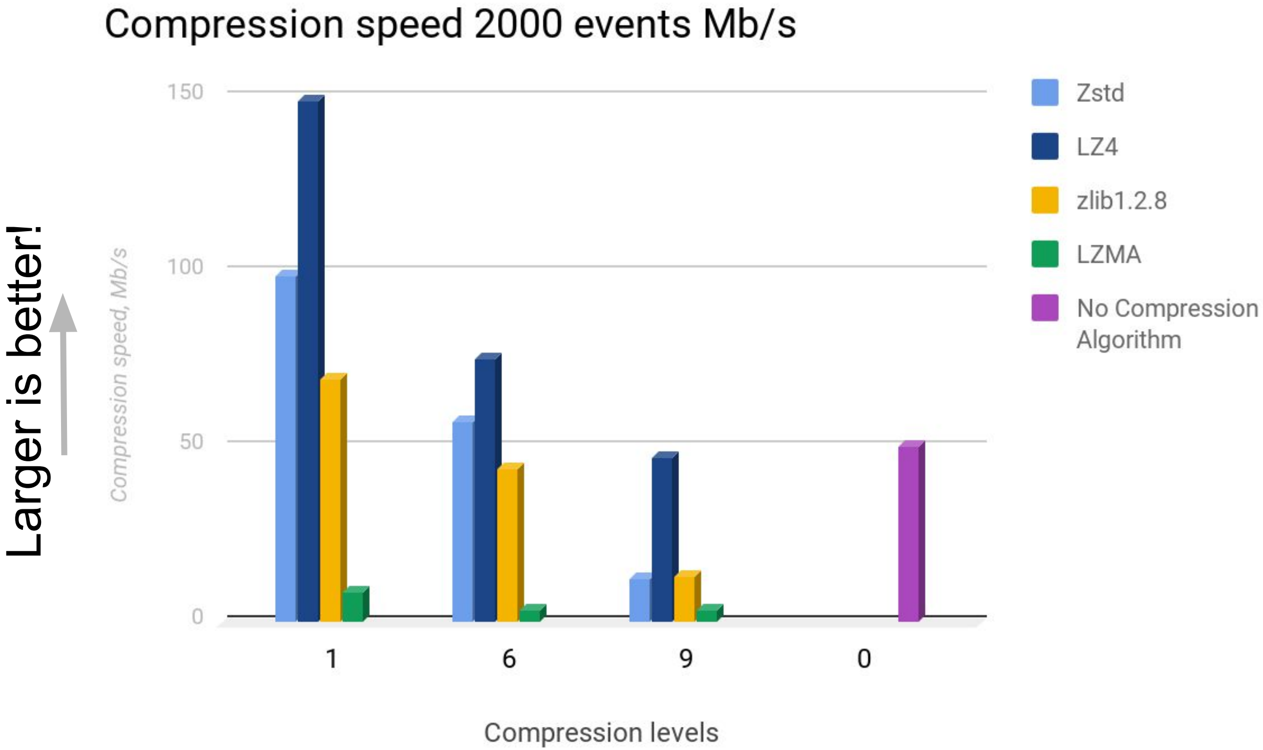
ZLIB-NG

- Fork of ZLIB, cleaning up and merging patches.
- Drop support of 16-bit platforms, ancient compilers
- Merged with all optimizations from Intel and Cloudflare. Supports more architectures than those forks.
- More actively developed.
- Check it out: https://github.com/Dead2/zlib-ng/tree/develop
    - Worth watching! Perhaps not enough history to make the jump yet…

Thank you for your attention!

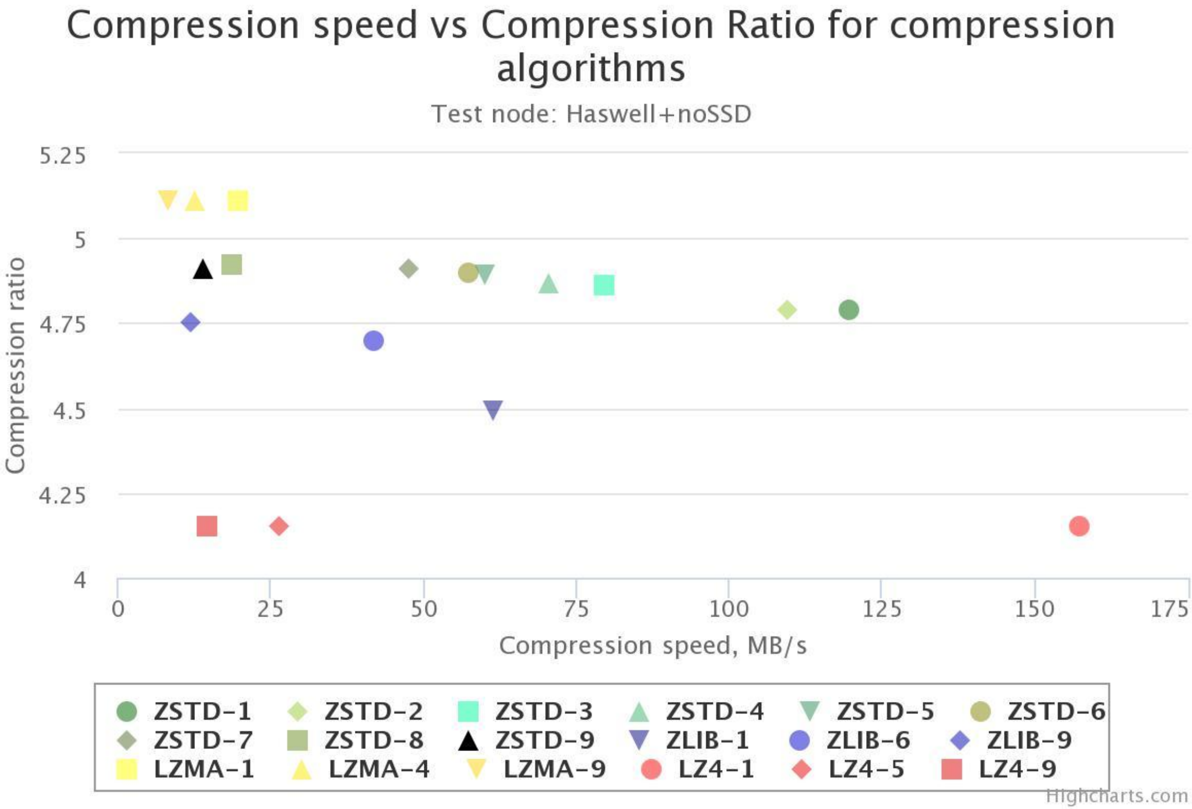
Backup Slides

Write Speed - Comparison Across Algorithms



ZSTD - Haswell x 56core - no SSD

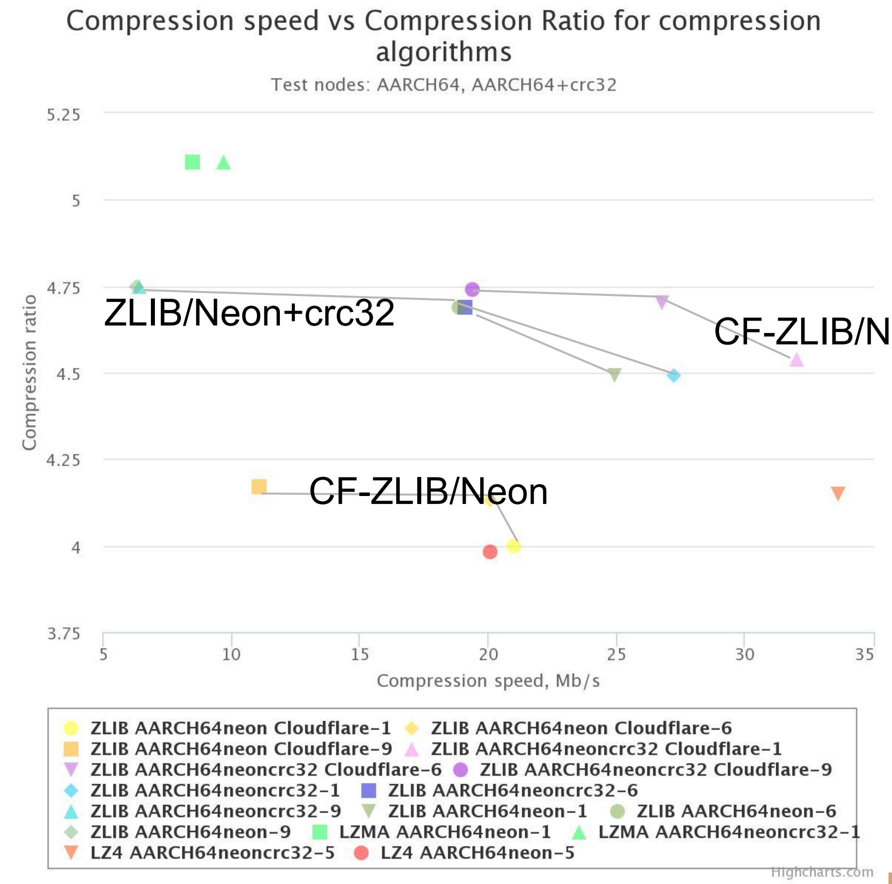
https://jsfiddle.net/oshadura/af6xt4n1/view



Larger is better

Cloudflare zlib vs zlib -AARCH64+CRC32 HiSilicon's Hi1612 processor (Taishan 2180)

http://jsfiddle.net/oshadura/qcwsx9y4/show



- LZ4 speed/compression ratio is almost on level of ZLIB Cloudflare speed/compression ratio (zlib-1)
- Improvement for zlib Cloudflare comparing to master for:
  - ZLIB-1/Neon+crc32: -31%
  - ZLIB-6/Neon+crc32: -36%
  - ZLIB-9/Neon +crc32-9: -69%
  - ZLIB-1/Neon: -10%
  - ZLIB-6/Neon: -10%
  - ZLIB-9/Neon: -50%