ZStandard in ZFS

Allan Jude -- allanjude@freebsd.org

Introduction

- 16 Years as FreeBSD Server Admin
- FreeBSD committer (ZFS, installer, boot loader, GELI [disk encryption], ucl, xo)
- FreeBSD Core Team (July 2016 - 2018)
- Co-Author of “FreeBSD Mastery: ZFS” and “FreeBSD Mastery: Advanced ZFS” with Michael W. Lucas
- Architect of the ScaleEngine Video CDN
- Host of BSDNow.tv Podcast
- Manage Over 1PB of ZFS storage across 30 locations around the globe

What is ZStandard

- New compression algorithm out of Facebook
- Created by Yann Collet, creator of LZ4
- Designed to provide gzip level compression but at much higher speeds
- Adaptive algorithm with multiple compression techniques: Finite State Entropy encoder, Huffman encoder
- 22 levels (speed & memory tradeoff)
- Dictionary Training

Ratio vs Speed Comparison (4.0GHz)

Compressor	Ratio	Compress	Decompress
Zstd 1.1.3 (-1)	**2.877**	**430 MB/s**	**1110 MB/s**
Zlib 1.2.8 (-1)	**2.743**	**110 MB/s**	**400 MB/s**
Brotli 0.5.2	2.708	400 MB/s	430 MB/s
Quicklz 1.5.0	2.238	550 MB/s	710 MB/s
Lzo1x 2.0.9	2.108	650 MB/s	830 MB/s
Lz4 1.7.5	**2.101**	**720 MB/s**	**3600 MB/s**
Snappy 1.1.3	2.091	500 MB/s	1650 MB/s
Lzf 3.6	2.077	400 MB/s	860 MB/s

Integration with ZFS

- ZFS has a very clean API to integrate additional compression algorithms
- ZSTD provides a mechanism to use your own memory allocator, with an opaque pointer for tracking. This fits the FreeBSD kernel memory allocator very nicely.
- Code Review Open:
- https://reviews.freebsd.org/D11124

Integration with FreeBSD

- Import ZSTD to contrib/zstd
- Has been upgraded a few times already
- Build lib/libzstd (libprivatezstd)
  - This library is only available to FreeBSD tools, any packages will depend on zstd from packages
- Modify zfs.ko to link against libprivatezstd
- It works!
- Integration with libstand so you can boot ZFS compressed with ZSTD
- Working on replacing gzip/bzip2 in loader for compressed kernel & mfsroot

Memory Management

- Currently an array of kmem_caches per major record size and compression level using ZSTD_estimateCCtxSize_advanced()
- Could use ZSTD_initStaticCCtx()
- Prototype uses multiple kmem caches to avoid consuming more memory than needed
- Decompression context is 152K

Record Size	zstd -1	zstd -3	zstd -19
16K	136K	200K	488K
128K	524K	1,004K	2,804K
1024K	556K	1,260K	13,556K
8192K	556K	1,260K	50,420K

How to Handle Levels?

- ZSTD has 19 (or 22 w/ ultra mode) levels.
- Adding all of these as unique compression types to the compress= property would eat up a lot of the limited namespace
- A new compress_level= property?
- Meaning would change with the compress= property, might be confusing to users
- How do you limit the acceptable values?
- Level 19 does not apply to gzip
- For now, prototype has:

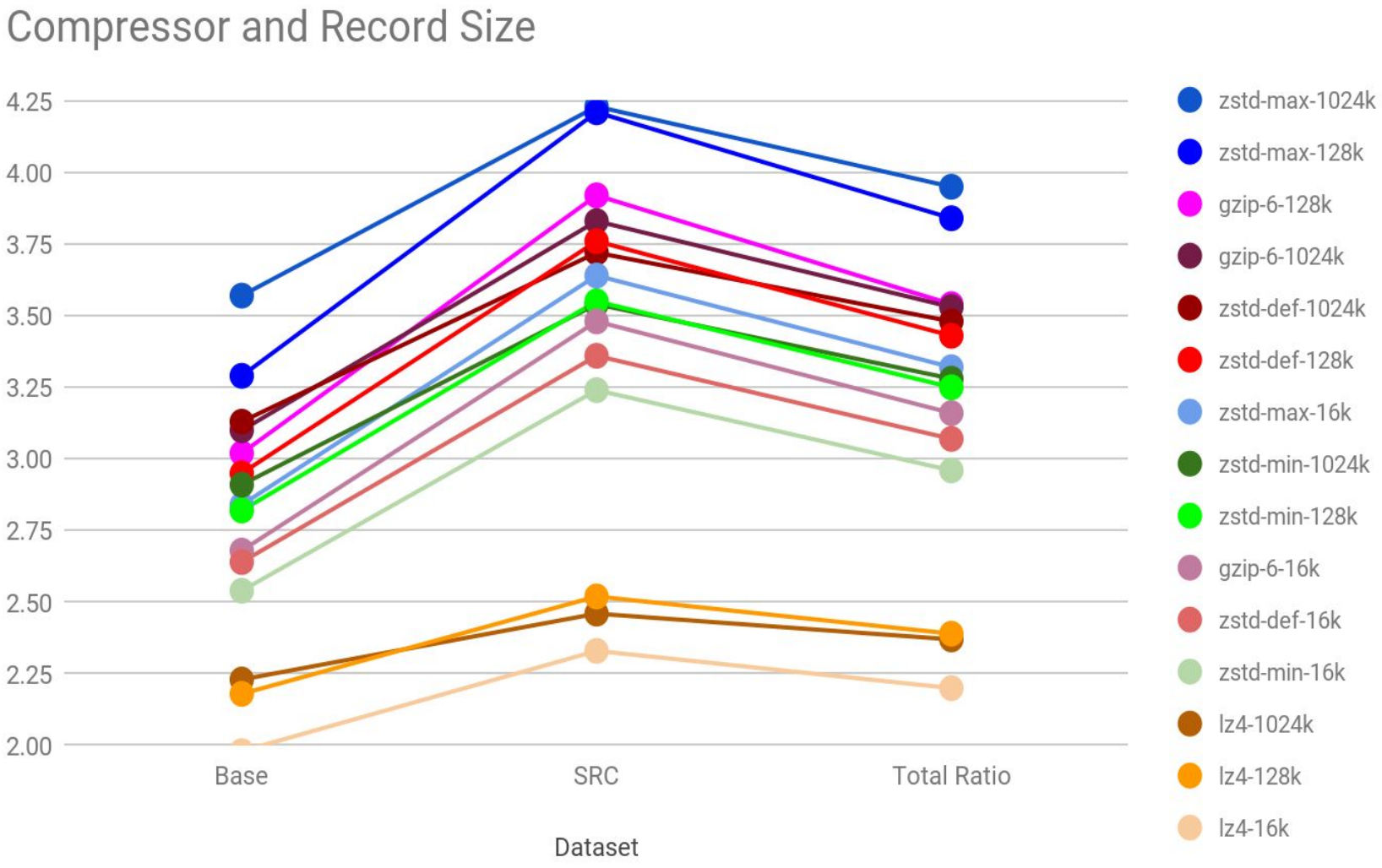
  zstd-min (1), zstd-default (3), zstd-max (19)

Level Comparison ZSTD (3.6GHz)

`zstd -b1 -e22 silesia_corpus`

Lvl	Ratio	Comp	Decomp
1	**2.877**	**335 MB/s**	**687 MB/s**
2	3.021	256 MB/s	650 MB/s
3	**3.164**	**195 MB/s**	**625 MB/s**
4	3.196	183 MB/s	625 MB/s
5	3.273	113 MB/s	612 MB/s
6	3.381	84.2 MB/s	642 MB/s
7	3.432	67.0 MB/s	657 MB/s
8	3.473	53.2 MB/s	670 MB/s
9	3.492	46.0 MB/s	674 MB/s
10	3.522	33.1 MB/s	682 MB/s
11	3.561	25.9 MB/s	678 MB/s
12	3.585	19.3 MB/s	686 MB/s
13	3.605	14.3 MB/s	691 MB/s
14	3.627	9.7 MB/s	691 MB/s
15	3.655	7.6 MB/s	704 MB/s
16	3.701	6.3 MB/s	700 MB/s
17	3.769	4.9 MB/s	700 MB/s
18	3.825	4.2 MB/s	700 MB/s
19	**3.923**	**3.3 MB/s**	**704 MB/s**
20	3.996	2.7 MB/s	585 MB/s
21	4.017	2.3 MB/s	585 MB/s
22	4.020	1.9 MB/s	585 MB/s
gz-1	2.743	77.7 MB/s	252 MB/s
gz-6	3.106	25.6 MB/s	265 MB/s
gz-9	3.133	10.8 MB/s	265 MB/s

Compress FreeBSD 11.1: 1.70 GB



Real World: Compressed Databases

- As EuroBSDCon 2017 in Paris last month, I did some on-the-spot performance analysis for a European payment processor
- They use a 128kb record size for their MySQL database, on purpose. The database is over 25TB all on SSD, so they rely on the high compression ratio to keep up with the demand for SSDs
- Write amplification is less of an issue since it is basically an append-only database

Our Pay-Per-View Database

- MySQL database is 14.2G uncompressed

Compression Algorithm	16K			128K			1024K		
	Size	Ratio	Time	Size	Ratio	Time	Size	Ratio	Time
lz4	3.76G	3.79x	0:55	2.90G	4.92x	0:54	2.81G	5.44	0:50
gzip-6	2.72G	5.27x	7:09	2.03G	7.02x	2:07	1.93G	7.91x	1:43
zstd-min	2.52G	5.68x	0:59	1.80G	7.94x	0:49	1.64G	9.33x	0:46
zstd-def	**2.49G**	**5.75x**	**1:04**	**1.77G**	**8.06x**	**0:51**	**1.58G**	**9.71x**	**0:50**
zstd-max	2.30G	6.24x	23:27	1.50G	9.52x	28:31	1.28G	11.99x	15:35

Dictionary Compression

- One of the major features of ZSTD is a special custom dictionary compression mode
- The idea is for compression of structured data, such as multiple JSON messages with the same key names. Train ZSTD with the template and get better and faster compression and decompression
- Would need some API to provide ZFS with 1 or more dictionaries per dataset
- Could this be used to compress arrays of block pointers? Or Indirect Blocks?

Compressed Record Size?

- Would it be practical to make a configuration where data is fed into the compression algorithm in chunks until the PHYSICAL record size is the desired value?
- What other impacts might this have on the system?
- Might this save wasted slack on 4Kn drives?
- What would an API look like that allowed you to continue to feed data in until the physical record was full?

ZSTD Adaptive Compression

- ZSTD has grown an adaptive compression feature, that automatically adjusts the compression level for maximum throughput on a throughput constrained output
- Typical use case: **`zfs send | compress | ssh | uncompress | zfs recv`**
- Likely not useful for individual blocks
- Could be combined with Nexenta “smartcompress” feature to get best compression without blocking

ZSTD APIs

- What new APIs might we want?
- Would ZFS benefit from using stream compression vs block compression?
- Some version of Early Abort like LZ4?
- Reduced memory modes for small blocks
- Does decompression context need to be > 150K if blocks are never more than 8M?
- More tuning and options for ZFS like workloads for small blocks (4k-16k record sizes for databases etc)
- ZSTD API that understand ABD / SGL

Start of the Project

- Aug 31 2016: ZSTD 1.0.0 released
- Sept 2016: I started integrating it into ZFS
- Found early on that ZSTD used some large stack variables and caused random seeming crashes (kernel stack overflow)
- Increases kstack_pages from 4 to 12 just to prove it will work before going forward
- Attempted to work around this by extending ‘HEAPMODE’ to malloc() the larger stack variables instead
- Early returns often made this is a bit messy

Timeline

- Oct 2016: Project stalled. Ifdef soup for HEAPMODE was a bit much
- Oct 2016: ZFS Developers summit conflicts with EuroBSDcon, I cannot attend
- Oct 2016: Saso Kiselkov works on ZSTD at ZFS Hackathon
- 2016: Nothing seems to comes of it
- Dec 14 2016: ZSTD 1.1.2 released with much reduced stack usage
- Jan 2017: FreeBSD Storage Summit rekindles interest in ZSTD in ZFS

Early Progress

- Update my working tree with newer ZSTD
- Resolve merge conflicts, remove most of HEAPMODE as it is no longer needed
- Solves most of the problems
- Build new ZFS kernel module and try it out
- Crashes with use-after-free or other memory corruption issues -- my fault
- ZSTD has custom memory allocator interface, so you can bring your own. Not used “everywhere” though. Trying to fix that did not go well.

Solution

- Replace few remaining ZSTD raw-malloc() calls with #ifdef _KERNEL to use kernel malloc (different prototype, extra arguments)
- Patch ends up relatively minor
- Talking with Yann Collet (ZSTD Author) about fixing this
- Yann is very interested in any other API requirements or suggests we have to better integrate with Kernel and ZFS

BSDNow.tv

- Weekly video podcast about the latest news in the BSD and IllumOS world
- Always looking for developers and active community members to interview
- Our archives are full of goodies:
  - Matt Ahrens
  - George Wilson
  - Bryan Cantrill
  - Adam Leventhal
  - Richard Yao
  - Alex Reese
  - Kirk McKusick
  - Josh Paetzel
  - Justin Gibbs
  - Paweł Jakub Dawidek
  - Sean Chittenden
  - 100+ Others