Integrating ZStandard into ZFS

Allan Jude -- allanjude@freebsd.org

Introduction

- 16 Years as FreeBSD Sysadmin
- FreeBSD committer (ZFS, installer, boot loader)
- FreeBSD Core Team (July 2016 - 2018)
- Co-Author of “FreeBSD Mastery: ZFS” and “FreeBSD Mastery: Advanced ZFS”
- Architect of the ScaleEngine Video CDN
- Host of BSDNow.tv Podcast
- Over 1PB of ZFS across 30 locations

What is ZStandard

- New compression algorithm out of Facebook
- Created by Yann Collet, author of LZ4
- Designed to beat gzip and be faster
- multiple compression techniques: Finite State Entropy coder, Huffman encoder
- 22 levels (speed & memory tradeoff)
  - New in 2018: Negative (faster) levels of compression
- Dictionary Training

Ratio vs Speed Comparison (4.0GHz)

Compressor	Ratio	Compress	Decompress
Zstd 1.3.4 (-1)	**2.877**	**470 MB/s**	**1380 MB/s**
Zlib 1.2.11 (-1)	**2.743**	**110 MB/s**	**400 MB/s**
Brotli 1.0.2	2.701	410 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 1.8.1	**2.101**	**750 MB/s**	**3700 MB/s**
Snappy 1.1.4	2.091	530 MB/s	1800 MB/s
Lzf 3.6	2.077	400 MB/s	860 MB/s

Start of the Project

- Aug 31 2016: ZSTD 1.0.0 released
- ZSTD used many large stack variables
- This caused seemingly random crashes (kernel stack overflow)
- PoC: Increases kstack_pages from 4 to 12
- Work Around: ‘HEAPMODE’, use malloc() for large stack variables
- Early returns often made this very messy

Timeline

- Oct 2016: Project stalled. #Ifdef soup for HEAPMODE was ugly and unmaintainable
- ZFS Dev summit conflicts with EuroBSDcon
- Saso Kiselkov works on ZSTD at Hackathon, Nothing seems to comes of it
- Dec: ZSTD 1.1.2 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
- Build new ZFS kernel module and try it out
- Crashes with use-after-free -- my fault
- ZSTD custom malloc interface, 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)
  - Eventually this was replaced with a macro
- Patch ends up relatively minor
- Talking with Yann Collet about fixing this
- Yann interested in any API suggests we have to better integrate with Kernel and ZFS

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 sys/contrib/zstd
- Has been upgraded a few times already
- The zstd command line tools are included in the FreeBSD base system for normal use
- Modify zfs.ko to borrow from libzstd-private
- Future: Integration with libstand (boot loader and related tools) so you can have ZFS boot pools compressed with ZSTD

Other Uses for ZSTD

- ZSTD is now a supported compressor for newsyslog(8), our log rotator
- ZSTD is now part of the FreeBSD kernel, used for compressed kernel crash dumps
- Working on replacing gzip & bzip2 in loader for compressed kernel & mfsroot (ramdisk)
- Maybe one day: ram or swap compression
- What other uses do you see for ZSTD?

Memory Management

- Currently an array of kmem_caches per major record size and compression level (avoid using a #19 kmem cache to compress #3)
- Could use ZSTD_initStaticCCtx() + locking?
- 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
- ZSTD has added unbounded *negative* levels
- Adding all of these as unique compression types to the compress= property would eat up the enum used in the block pointer
- Discussed at the OpenZFS Developer Summit 2017 with rm@ and skiselkov@

Solution for Levels

- A new hidden zstd_complevel= property
- User still does: zfs set compress=zstd-7
- As it crosses the IOCTL boundary, it is split into: compress=zstd + zstd_complevel=7
- Put back together on the way back to userspace, so ‘zfs get’ displays as expected
- Block Pointer only needs to know which decompression function to use, not level

Further Challenges

- Matt reviews prototype, spots a problem!
- Compressed ARC disabled + L2ARC= no go
- Writes to L2ARC must be recompressed to have same checksum as the on-disk BP
- This requires knowing the compression level that was used, but the BP only says ‘zstd’
- Now we must store the compression level on disk, in the top 6 bits of the logical size

Becoming a Yak Farmer

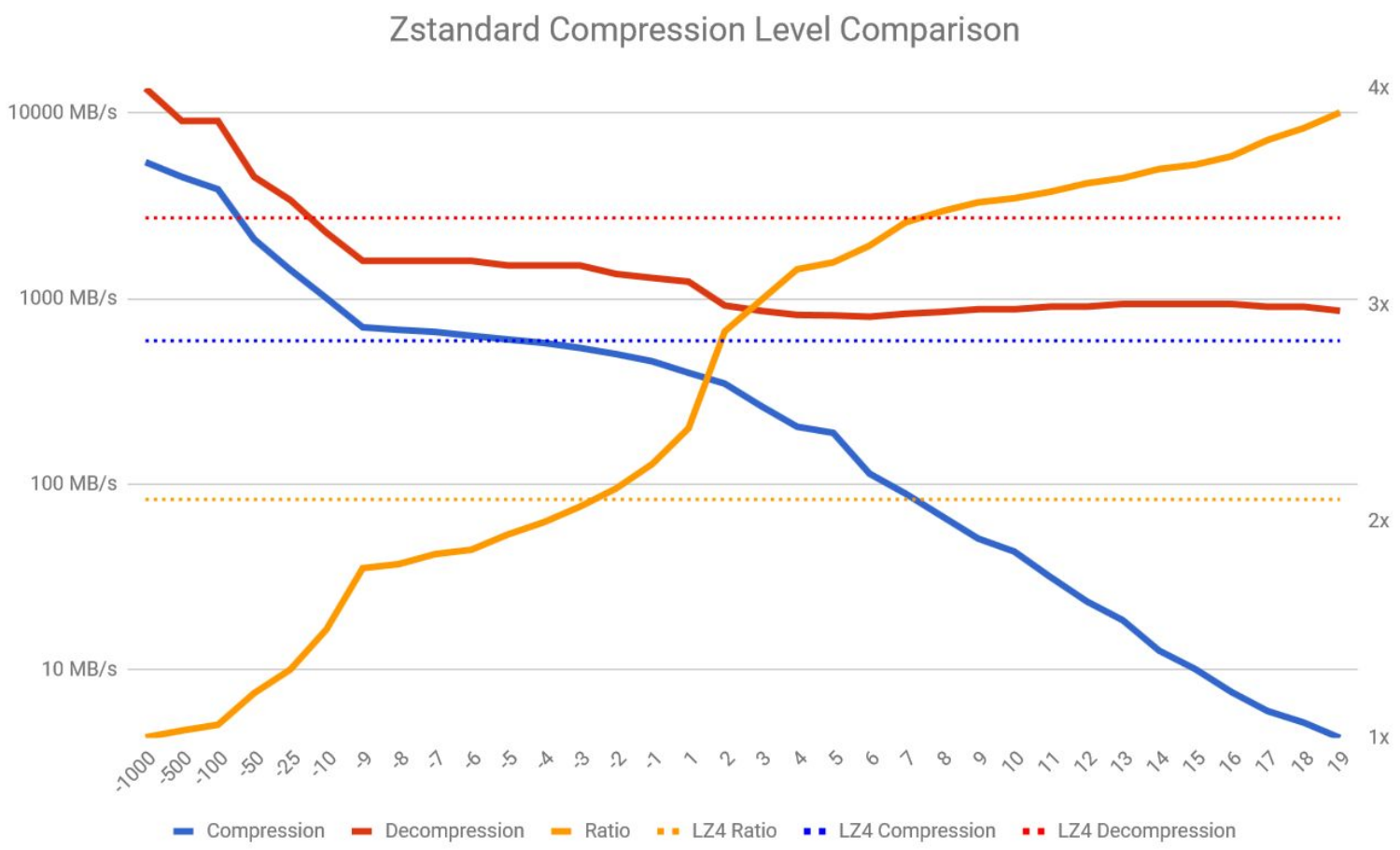
- FreeBSD NUMA improvements
  - A bug caused the ARC to constantly try to free itself
- 8856 arc_cksum_is_equal() doesn't take into account ABD-logic
  - Fixed 2 days before I found it, an hour after my pull
- 9321 arc_loaned_bytes can underflow
  - No one had ever zfs recv'd a compressed stream while having compressed ARC disabled?
  - My patch has been merged upstream

Level Comparison ZSTD (3.6GHz)

zstd -b --fast=8 -e19 silesia.txt

Lvl	Ratio	Comp	Decomp
Zstd-8	1.849	661 MB/s	1595 MB/s
Zstd-4	2.068	542 MB/s	1427 MB/s
Lz4 1	2.101	592 MB/s	2716 MB/s
Zstd-3	2.152	511 MB/s	1427 MB/s
Zstd-1	2.430	400 MB/s	1233 MB/s
Lz4 3	2.606	90.2 MB/s	2553 MB/s
gzip1	2.743	83.1 MB/s	246 MB/s
Zstd 1	2.877	348 MB/s	917 MB/s
Zstd 2	3.021	264 MB/s	858 MB/s
gzip6	3.106	28.9 MB/s	260 MB/s
gzip9	3.133	11.9 MB/s	265 MB/s
Zstd 3	3.164	204 MB/s	816 MB/s
Zstd 4	3.196	189 MB/s	811 MB/s
Zstd 5	3.273	114 MB/s	798 MB/s

Lvl	Ratio	Comp	Decomp
Zstd 6	3.381	88.9 MB/s	828 MB/s
Zstd 7	3.432	67.3 MB/s	847 MB/s
Zstd 8	3.473	51.0 MB/s	875 MB/s
Zstd 9	3.492	43.4 MB/s	875 MB/s
Zstd 10	3.522	31.6 MB/s	904 MB/s
Zstd 11	3.561	23.4 MB/s	904 MB/s
Zstd 12	3.585	18.5 MB/s	935 MB/s
Zstd 13	3.627	12.7 MB/s	935 MB/s
Zstd 14	3.647	10.1 MB/s	935 MB/s
Zstd 15	3.686	7.6 MB/s	935 MB/s
Zstd 16	3.761	6.0 MB/s	903 MB/s
Zstd 17	3.816	5.2 MB/s	903 MB/s
Zstd 18	3.888	4.3 MB/s	858 MB/s
Zstd 19	3.926	3.7 MB/s	853 MB/s


Zstandard Compression Level Comparison
10000 MB/s
1000 MB/s
100 MB/s
10 MB/s
4x
3x
2x
1x
-1000
-500
-100
-50
-25
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Compression
Decompression
Ratio
LZ4 Ratio
LZ4 Compression
LZ4 Decompression

Real World: Compressed Databases

- Last fall, I was doing some performance analysis for a European payment processor
- They use 128kb record size for their MySQL database. The database is over 25TB, all on SSDs, they rely on high compression ratios 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 (2.6GHz)

MySQL database: 45.9G uncompressed

Algorithm	16K				128K				1024K			
	Size	Ratio	Rate	Time	Size	Ratio	Rate	Time	Size	Ratio	Rate	Time
lz4	19.3G	2.23x	58.9	12:29	9.50G	4.56x	145	5:04	8.29G	5.53x	182	4:18
gzip-6	15.4G	2.81x	49.4	14:51	7.38G	5.87x	75.6	9:46	6.34G	7.23x	82.8	9:27
zstd-fast-1**	-	-	-	-	-	-	-	-	*4.48G*	*10.22x*	*381*	*2:03*
zstd-1	13.8G	3.14x	60.8	11:57	6.08G	7.13x	160	4:37	4.35G	10.55x	208	3:46
zstd-3	**12.8G**	**3.38x**	**53.3**	**13:47**	**5.86G**	**7.40x**	**136**	**5:24**	**4.16G**	**11.02x**	**176**	**4:26**
zstd-10	11.9G	3.65x	22.4	32:51	5.60G	7.74x	26.5	27:55	3.93G	11.67x	25.2	31:05
zstd-19	11.6G	3.73x	13.0	56:23	5.38G	8.07x	10.2	72:13	3.64G	12.60x	8.6	90:48

** Estimated, negative levels are not yet supported

Dictionary Compression

- ZSTD has a special custom dictionary mode
- Designed for compression of structured data, such as multiple JSON messages with the same key names. Train ZSTD with the template and compresses/decompress better and faster
- Would need some API to provide ZFS with 1 or more dictionaries per dataset (like crypto keys)
- Could this be used to compress arrays of block pointers? Or Indirect Blocks?

ZSTD Adaptive Compression

- ZSTD has grown an adaptive compression feature, automatically adjusts the level for maximum throughput on constrained pipe
- Typical use case: zfs send|zstd|ssh|unzstd|zfs recv
- Change level based on volume of dirty data?
- Could be combined with Nexenta “smart compress” feature to “learn” about a file and get best compression without blocking

ZSTD APIs

- What new APIs might we want?
- stream compression vs block compression?
- Reduced memory modes for small blocks
- Does decompression context need to be > 150K if blocks are never more than 8M?
- More tuning for small blocks 4k-16k records
- ZSTD API that understand ABD / SGL

BSDNow.tv

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