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)

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Ratio</th>
<th>Compress</th>
<th>Decompress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zstd 1.3.4 (-1)</td>
<td>2.877</td>
<td>470 MB/s</td>
<td>1380 MB/s</td>
</tr>
<tr>
<td>Zlib 1.2.11 (-1)</td>
<td>2.743</td>
<td>110 MB/s</td>
<td>400 MB/s</td>
</tr>
<tr>
<td>Brotli 1.0.2</td>
<td>2.701</td>
<td>410 MB/s</td>
<td>430 MB/s</td>
</tr>
<tr>
<td>Quicklz 1.5.0</td>
<td>2.238</td>
<td>550 MB/s</td>
<td>710 MB/s</td>
</tr>
<tr>
<td>Lzo1x 2.0.9</td>
<td>2.108</td>
<td>650 MB/s</td>
<td>830 MB/s</td>
</tr>
<tr>
<td>Lz4 1.8.1</td>
<td>2.101</td>
<td>750 MB/s</td>
<td>3700 MB/s</td>
</tr>
<tr>
<td>Snappy 1.1.4</td>
<td>2.091</td>
<td>530 MB/s</td>
<td>1800 MB/s</td>
</tr>
<tr>
<td>Lzf 3.6</td>
<td>2.077</td>
<td>400 MB/s</td>
<td>860 MB/s</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Record Size</th>
<th>zstd 1</th>
<th>zstd 3</th>
<th>zstd 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>16K</td>
<td>136K</td>
<td>200K</td>
<td>488K</td>
</tr>
<tr>
<td>128K</td>
<td>524K</td>
<td>1,004K</td>
<td>2,804K</td>
</tr>
<tr>
<td>1024K</td>
<td>556K</td>
<td>1,260K</td>
<td>13,556K</td>
</tr>
<tr>
<td>8192K</td>
<td>556K</td>
<td>1,260K</td>
<td>50,420K</td>
</tr>
</tbody>
</table>
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
```

<table>
<thead>
<tr>
<th>Lvl</th>
<th>Ratio</th>
<th>Comp</th>
<th>Decomp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zstd-8</td>
<td>1.849</td>
<td>661 MB/s</td>
<td>1595 MB/s</td>
</tr>
<tr>
<td>Zstd-4</td>
<td>2.068</td>
<td>542 MB/s</td>
<td>1427 MB/s</td>
</tr>
<tr>
<td>Lz4 1</td>
<td>2.101</td>
<td>592 MB/s</td>
<td>2716 MB/s</td>
</tr>
<tr>
<td>Zstd-3</td>
<td>2.152</td>
<td>511 MB/s</td>
<td>1427 MB/s</td>
</tr>
<tr>
<td>Zstd-1</td>
<td>2.430</td>
<td>400 MB/s</td>
<td>1233 MB/s</td>
</tr>
<tr>
<td>Lz4 3</td>
<td>2.606</td>
<td>90.2 MB/s</td>
<td>2553 MB/s</td>
</tr>
<tr>
<td>gzip1</td>
<td>2.743</td>
<td>83.1 MB/s</td>
<td>246 MB/s</td>
</tr>
<tr>
<td>Zstd 1</td>
<td>2.877</td>
<td>348 MB/s</td>
<td>917 MB/s</td>
</tr>
<tr>
<td>Zstd 2</td>
<td>3.021</td>
<td>264 MB/s</td>
<td>858 MB/s</td>
</tr>
<tr>
<td>gzip6</td>
<td>3.106</td>
<td>28.9 MB/s</td>
<td>260 MB/s</td>
</tr>
<tr>
<td>gzip9</td>
<td>3.133</td>
<td>11.9 MB/s</td>
<td>265 MB/s</td>
</tr>
<tr>
<td>Zstd 3</td>
<td>3.164</td>
<td>204 MB/s</td>
<td>816 MB/s</td>
</tr>
<tr>
<td>Zstd 4</td>
<td>3.196</td>
<td>189 MB/s</td>
<td>811 MB/s</td>
</tr>
<tr>
<td>Zstd 5</td>
<td>3.273</td>
<td>114 MB/s</td>
<td>798 MB/s</td>
</tr>
<tr>
<td>Zstd 6</td>
<td>3.381</td>
<td>88.9 MB/s</td>
<td>828 MB/s</td>
</tr>
<tr>
<td>Zstd 7</td>
<td>3.432</td>
<td>67.3 MB/s</td>
<td>847 MB/s</td>
</tr>
<tr>
<td>Zstd 8</td>
<td>3.473</td>
<td>51.0 MB/s</td>
<td>875 MB/s</td>
</tr>
<tr>
<td>Zstd 9</td>
<td>3.492</td>
<td>43.4 MB/s</td>
<td>875 MB/s</td>
</tr>
<tr>
<td>Zstd 10</td>
<td>3.522</td>
<td>31.6 MB/s</td>
<td>904 MB/s</td>
</tr>
<tr>
<td>Zstd 11</td>
<td>3.561</td>
<td>23.4 MB/s</td>
<td>904 MB/s</td>
</tr>
<tr>
<td>Zstd 12</td>
<td>3.585</td>
<td>18.5 MB/s</td>
<td>935 MB/s</td>
</tr>
<tr>
<td>Zstd 13</td>
<td>3.627</td>
<td>12.7 MB/s</td>
<td>935 MB/s</td>
</tr>
<tr>
<td>Zstd 14</td>
<td>3.647</td>
<td>10.1 MB/s</td>
<td>935 MB/s</td>
</tr>
<tr>
<td>Zstd 15</td>
<td>3.686</td>
<td>7.6 MB/s</td>
<td>935 MB/s</td>
</tr>
<tr>
<td>Zstd 16</td>
<td>3.761</td>
<td>6.0 MB/s</td>
<td>903 MB/s</td>
</tr>
<tr>
<td>Zstd 17</td>
<td>3.816</td>
<td>5.2 MB/s</td>
<td>903 MB/s</td>
</tr>
<tr>
<td>Zstd 18</td>
<td>3.888</td>
<td>4.3 MB/s</td>
<td>858 MB/s</td>
</tr>
<tr>
<td>Zstd 19</td>
<td>3.926</td>
<td>3.7 MB/s</td>
<td>853 MB/s</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>16K</th>
<th>128K</th>
<th>1024K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Ratio</td>
<td>Rate</td>
</tr>
<tr>
<td>lz4</td>
<td>19.3G</td>
<td>2.23x</td>
<td>58.9</td>
</tr>
<tr>
<td>gzip-6</td>
<td>15.4G</td>
<td>2.81x</td>
<td>49.4</td>
</tr>
<tr>
<td>zstd-fast-1**</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>zstd-1</td>
<td>13.8G</td>
<td>3.14x</td>
<td>60.8</td>
</tr>
<tr>
<td>zstd-3</td>
<td>12.8G</td>
<td>3.38x</td>
<td>53.3</td>
</tr>
<tr>
<td>zstd-10</td>
<td>11.9G</td>
<td>3.65x</td>
<td>22.4</td>
</tr>
<tr>
<td>zstd-19</td>
<td>11.6G</td>
<td>3.73x</td>
<td>13.0</td>
</tr>
</tbody>
</table>

** 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