NoLoad Filesystem: A Stacked Filesystem for NVMe Computational Storage

David Sloan, Logan Gunthorpe, Stephen Bates
Topics

- Introduction: The case for compression
- NoLoad NVMe-Based CSx
- Transparent Compression using an NVMe CSx
- Stacked Filesystems
- NoLoad Filesystem
  - Compression Method
  - Storage Architecture
  - Interface
  - Performance
Introduction
Introduction: The Case For Compression

- Data volumes are exploding
- NAND is getting cheaper but not as cheap as HDDs.
- NAND provides x1000 the performance of HDDs wrt IOPS, throughput and latency
- Compression can bridge the cost gap
- But it has to be performant, efficient and easy to consume!
NoLoad
An NVMe-Based Computational Storage Device
NoLoad® Computational Storage Device (CSx)

Eideticom’s NoLoad® CSx
Purpose built for acceleration of storage and compute-intensive workloads

- **NoLoad Software Stack**
  - End-to-end computational storage solution providing transparent computational offload
  - Complete Software and IP core stack

- **NoLoad NVMe Front End**
  - NVMe compliant, standards-based interface
  - High performance interface tuned for computation

- **NoLoad Computational Accelerators**
  - Storage Accelerators: Compression, Encryption, Erasure Coding, Deduplication
  - Compute Accelerators: Data Analytics, Video Codec, AI and ML

World’s First Computational Storage Processor (certified by UNH-IOL)
SNIA Computational Storage Terminology

**Computational Storage Processor (CSP)**
A component that provides computational services to a storage system without providing persistent storage

**Computational Storage Drive (CSD)**
A component that provides persistent data storage and computational services

NoLoad on Xilinx Alveo U50

NoLoad on Samsung SmartSSD
Transparent Compression using an NVMe CSx
Transparent Compression

- Transparent compression is compression that the application is unaware of.
- This can happen in one of four places:
  - On the device (multiple vendors)
  - In the block layer (e.g. VDO)
  - In the filesystem (e.g. ZFS)
  - In a stacked filesystem (e.g. this work)
CSx as an Accelerator

Application

Access Library

NVMe Driver

NVMe CSx

NVMe Storage
Transparent Computation

Application
Storage IO

Data Transform
- Compression
- Encryption
- ECC

Fn()
Fn⁻¹()

NVMe Storage
Transparent Compression
Non-Transparent Compression

Data $\rightarrow$ write(...) $\rightarrow$ NVMe $\rightarrow$ Data

... $\rightarrow$ read(...) $\rightarrow$ NVMe $\rightarrow$ Data

Data $\rightarrow$ csx_fn(...) $\rightarrow$ Buffer $\rightarrow$ Data

write(...) $\rightarrow$ NVMe $\rightarrow$ read(...) $\rightarrow$ Buffer

NVMe $\rightarrow$ csx_inv(...) $\rightarrow$ Data

13 | 2020 Storage Developer Conference. © Eidetic Communications Inc. All Rights Reserved.
Transparent Compression

Application

Kernel

System read(...)

System write(...)

NVMe CSx

NVMe Storage
Stacked Filesystems
Stacked Filesystems

- Provide a code injection point between applications and the filesystem
- Adds functionality to existing filesystems
- Easy to integrate into existing SW stacks!
Stacked Filesystem

```
Stacked filesystem
e.g.
mount -t nlfs /mount/lower /mount/stacked
```

```
Lower Filesystem
e.g.
mkfs.xfs /dev/nvme0n1p1
mount /dev/nvme0n1p1 /mount/lower
```
Stacked Filesystems

- Applications
  - NFS
    - NoLoad FS
      - XFS
    - NoLoad FS
      - XFS
    - NoLoad FS
      - EXT4
  - VFS
    - Disk
    - Disk
    - Disk
NoLoad Filesystem

- Application
- User | Kernel

- Filesystem
  - NoLoad Filesystem
  - XFS, EXT4, etc.

- NVMe Driver
  - NVMe Storage
  - Accelerator
  - NoLoad CSP
  - NoLoad CSD

- NVMe Storage
NoLoad Filesystem

Applications

User

Kernel

NoLoad Filesystem

Pass Through
stat(...)
mkdir(...)
...

xattr
getxattr(...)
setxattr(...)
...

IO ops
read(...)
write(...)
...

Lower Filesystem

Compute State

NVMe CSx
NoLoad Filesystem

NoLoad Filesystem

Thin IO Check

File ops

Has nlsf_xattr?

Yes

No

Pass Through

NoLoad FS Core

xattr
getxattr(...) setxattr(...) ...

IO ops
read(...) write(...) ...

Compute State

Lower Filesystem

NVMe CSx
NoLoad FS: Compression Method
NLFS: Compressed Filesystem

- Use existing infrastructure, Don’t reinvent the wheel
- SW Compatible
  - Z-Lib encoding
- Record-based
ZLib-Encoding

- SW Recovery
- Asymmetric topologies
- Use Existing Kernel Infrastructure
NoLoad CSP

Deflate

Data Buffers

Data In

Data Out

Lower FS Write

NVMe Write

Compress Job

NVMe Write

NVMe Read

Compression Core
NoLoad CSD

Deflate

Data Buffers

Data In

Compress Job

NVMe Write

NVMe Read

Compression Core

Lower FS Write

NVMe Write

CMB

Data Out

NVMe Storage

NVMe Write

NVMe Storage

NVMe Read

Data In

NoLoad CSD
NoLoad FS: Storage Architecture
Storing Compressed Data

- Hole punching informs lower fs where data can be omitted
Compression Records

Z-Lib Compression

Data Record

Compressed Data

- 1 MiB -> 7 kiB rounded up to 8 kiB

Lower FS

Block 0 | Block 1 | Block 2 | ... | Block n

Stored Data | Hole | Hole
Compression Records

- Low compressibility data is stored raw to improve read performance
Compression Records

NLFS File

<table>
<thead>
<tr>
<th>Record 0</th>
<th>Record 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Dat</td>
<td>Hole</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Record 2</td>
<td>Record 3</td>
</tr>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Blk</td>
<td>Blk</td>
</tr>
<tr>
<td>Data</td>
<td>Raw Data</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NoLoad FS: Interface
System Stats

- Report compression info using built in tools
  - `ls`'s `-s` option displays stored size, which can be less than the file size

```
$ ls -lhs /nlfs/
total 128
128K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 10:58 rand_ac.dat
```

Size on Disk | File Size
--- | ---
128K | 1.0M
## Implementation Specific Stats

<table>
<thead>
<tr>
<th>Stored Size</th>
<th>Record Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>8129.4340.65536.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>00f0</td>
<td>8129.4340.65536.</td>
</tr>
</tbody>
</table>

- Allows the reporting and modification of implementation specific settings on a per-file basis

```bash
$xattr -l rand_ac.dat
user.nlfs_recordsize: 65536
user.nlfs_recordstat:
0000 ... 8129.4340.65536.
... 00f0 ... 8129.4340.65536.
```
### NLFS Debug Info

- **compress_ratio**
  - Compression ratio of mount as a whole (file size/size on disk)
- **[lower|mnt]_path**
  - Paths of lower fs and nlfs mount respectively

```bash
$ ls /sys/kernel/debug/nlfs/0
compress_records_actions_zlib  compress_ratio
compress_records_actions_nvme  compress_records_write
compress_records_actions_read  compress_records_read
decompress_records_actions_zlib lower_path  raw_records_read
          mnt_path  raw_records_write
```

**NLFS Mount ID**
NLFS Debug Info

$ls /sys/kernel/debug/nlfs/0

- \texttt{compress\_records\_actions\_nvme}
- \texttt{decompress\_records\_actions\_nvme}
- \texttt{compress\_records\_actions\_zlib}
- \texttt{decompress\_records\_actions\_zlib}
- \texttt{compress\_ratio}
- \texttt{compress\_records\_write}
- \texttt{compress\_records\_read}
- \texttt{lower\_path}
- \texttt{raw\_records\_read}
- \texttt{mnt\_path}
- \texttt{raw\_records\_write}

- \textbf{[de]compress\_records\_actions\_x}
  - Number of actions taken for record type
    - Records processed using CSx
    - Records processed using SW
### NLFS Debug Info

- `compress_records_x`
  - Number of compressed records read/written to disk
- `raw_records_x`
  - Number of raw records read/written to disk

```bash
$ ls /sys/kernel/debug/nlfs/0
compress_records_actions_zlib    compress_ratio
compress_records_actions_nvme    compress_records_write
compress_records_actions_read    compress_records_read
decompress_records_actions_zlib  lower_path  raw_records_read
                                 mnt_path    raw_records_write
```
Usage Example

```
$ls /xfs/
raw.dat

$mount -t nlfs /xfs /nlfs -o recordsize=1M
$touch /nlfs/sequential.dat
$touch /nlfs/rand_ac.dat

$xattr -w user.nlfs_recordsize 65536 /nlfs/rand_ac.dat
$cp /nlfs/raw.dat /nlfs/sequential.dat
$cp /nlfs/raw.dat /nlfs/rand_ac.dat
$sync

$ls -lhs /nlfs/
total 1.2M
128K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:23 rand_ac.dat
8.0K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:23 sequential.dat
1.0M -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:22 raw.dat
```
NoLoad FS: Performance
**Configuration**

**User**

**Kernel**

**FIO**

**NoLoad Filesystem**

**XFS Filesystem**

**8xNVMe SSD Raid-0**

**NVMe Storage**

**NVMe Storage**

**NoLoad CSP**

**Compression**

**Decompression**

**Host Machine**

AMD EPYC 7302P @ 3 GHz
128 GB DDR4 @ 2933 MT/s
Write Throughput vs Record Size

Sequential Writes

PCIe Gen3x16

Throughput (GB/s)

0% Cmp 30% Cmp 70% Cmp 90% Cmp

1 MiB 256 kiB 64 kiB
Read Throughput vs Record Size

Sequential Reads

<table>
<thead>
<tr>
<th>Throughput (GB/s)</th>
<th>0% Cmp</th>
<th>30% Cmp</th>
<th>70% Cmp</th>
<th>90% Cmp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MiB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 kiB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 kiB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PCIe Gen3x16
Write Latency vs Record Size

Random Writes

Latency (µs)

- Raw XFS
- 0% Cmp
- 30% Cmp
- 70% Cmp
- 90% Cmp

Record Sizes:
- 256 kiB
- 64 kiB
Read Latency vs Record Size

Latency (µs)

Random Reads

- Raw XFS
- 0% Cmp
- 30% Cmp
- 70% Cmp
- 90% Cmp

256 kiB
64 kiB
Conclusions

- Data volumes are exploding, and NAND is getting cheaper. Hardware accelerated compression can help bridge the gap.
- NVMe CSx devices allow high speed injection of HW accelerated data transforms into applications.
- Stacked filesystems allow for greater configuration flexibility and leverage existing filesystem optimizations.
- A Stacked filesystem can be inserted into existing infrastructure with minimal effort.
Please take a moment to rate this session.

Your feedback matters to us.