The Rise of Computational Storage

Homogeneous Computing → Heterogeneous Computing

Compute
FPGA/GPU/TPU
End of Moore’s Law

Networking
SmartNICs
10 → 100-400Gb/s

Storage
Computational Storage
Fast & Big Data Growth

Domain Specific Compute
Computational Storage: A Simple Idea

- End of Moore’s Law $\Rightarrow$ heterogeneous computing

Low-hanging fruits

Computational Storage Drive (CSD) with Data Path Transparent Compression
Data Compression

One block (e.g., 8KB, 32KB, or 128KB) → Compression → Compressed block

- Improve compression ratio & reduce storage space waste
- 4KB-aligned
- Wasted
- Block size → R/W amplification → Performance
- lz4 → zlib/ZSTD → CPU overhead
Transparent Compression

Journaling filesystem (ext4, XFS, ...)

Block layer w/ transparent compression (VDO, ...)

4KB

4KB sector

A B C D E

Per-4KB lz4 Compression

Improve compression ratio & reduce storage space waste

Iz4 ➔ zlib/ZSTD ➔ CPU overhead ➔ Performance
Transparent Compression

Journaling filesystem (ext4, XFS, ...)

Normal block layer

Computational Storage Drive (CSD) with Data Path Transparent Compression
ScaleFlux Computational Storage Drive: CSD 2000

- Complete, validated solution
  - Pre-Programmed FPGA
  - Hardware
  - Software
  - Firmware
- No FPGA knowledge or coding
- Field upgradeable
- Standard U.2 & AIC form factors

Multiple, discrete components for Compute and SSD Functions

Single FPGA combines Compute and SSD Functions
CSD 2000: Data Path Transparent Compression

Block-layer transparent compression (Linux VDO, VMware, ...)

ScaleFlux CSD 2000: Data path compression

HW zlib compression  Much higher compression ratio @ zero CPU cost  Tight packing  Zero space waste
CSD 2000: Compression Without Compromise

Data Size Relative to Uncompressed (4KB Aligned)

- No Compression
- LZ4 1.9.2
- zstd 1.4.3-1
- zstd 1.4.3-6
- zlib 1.2.11-1
- zlib 1.2.11-6
- CSD 2000 Compression

- Half the Space
- No CPU Overhead
- No App Latency

Achieve Greater Storage Space Savings with CSD 2000

Using Canterbury Corpus files (2,810,874B uncompressed); 8KB compression block to simulate relational databases (e.g. PostgreSQL, OracleDB, SQL Server)
CSD 2000: Data Path Compression/Decompression

Increasing Mix R/W Performance Advantage with Larger Block Sizes
Straightforward Usage in Postgres

- Xeon E5-2667 v4 32-core @ 3.2GHz, 256GB DRAM
- CentOS 7.5.1804, Postgres 10.10, Sysbench 1.1.0 (64 threads)
- 3.2TB vendor-A NVMe drive vs. 3.2TB ScaleFlux drive
- **Transparent compression:** 2TB Postgres dataset → 776GB (60% reduction)

**Where does the TPS gain on read-intensive workloads come from?**

- Smaller storage footprint
- Less flash memory access conflict
- Higher page read throughput

 normalized TPS
(higher is better)

- read_only
- update_non_index
- update_index
- point_select
- read_write

Vendor-A Drive
ScaleFlux Drive
One Step Further

- **8KB/page**
  - Data
  - Fillfactor (FF)

- **Reserved for future update**

- **Performance**
  - FF down
  - Storage space up

- **Normalized Performance**
  - 0’s
  - Compressed data

- **Data path compression**

- **Commodity NVMe**
- **SFX NVMe**

- **Physical storage usage**
  - 300GB
  - 600GB
  - 1,200GB
One Step Further (Sysbench-TPCC)

<table>
<thead>
<tr>
<th>Fillfactor</th>
<th>Drive</th>
<th>Logical size (GB)</th>
<th>Physical size (GB)</th>
<th>Comp Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Vendor-A CSD 2000</td>
<td>740</td>
<td>740</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>178</td>
<td></td>
<td>4.12</td>
</tr>
<tr>
<td>75</td>
<td>Vendor-A CSD 2000</td>
<td>905</td>
<td>905</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>189</td>
<td></td>
<td>4.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fillfactor</th>
<th>Drive</th>
<th>Logical size (GB)</th>
<th>Physical size (GB)</th>
<th>Comp Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Vendor-A CSD 2000</td>
<td>1,433</td>
<td>1,433</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.19</td>
</tr>
<tr>
<td>75</td>
<td>Vendor-A CSD 2000</td>
<td>1,762</td>
<td>1,762</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.82</td>
</tr>
</tbody>
</table>

Normalized TPS

Vendor-A

CSD 2000

Fillfactor | Drive   | Logical size (GB) | Physical size (GB) | Comp Ratio |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Vendor-A CSD 2000</td>
<td>740</td>
<td>740</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>178</td>
<td></td>
<td>4.12</td>
</tr>
<tr>
<td>75</td>
<td>Vendor-A CSD 2000</td>
<td>905</td>
<td>905</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>189</td>
<td></td>
<td>4.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fillfactor</th>
<th>Drive</th>
<th>Logical size (GB)</th>
<th>Physical size (GB)</th>
<th>Comp Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Vendor-A CSD 2000</td>
<td>1,433</td>
<td>1,433</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.19</td>
</tr>
<tr>
<td>75</td>
<td>Vendor-A CSD 2000</td>
<td>1,762</td>
<td>1,762</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.82</td>
</tr>
</tbody>
</table>
- Cycling endurance is a big issue in the upcoming QLC NAND flash memory
- B-tree is well-known for its large write amplification under small record size
CSD 2000: Accessing Extended Capacity

**File System or Raw Device**
- Standard Linux framework
- API for gathering current free space
- Integrate with server monitoring tools
- Requires “mount with discard” for FS
- Active storage capacity monitoring by the application/admin

**File System**
- Included in ScaleFlux Tools
- Runs as a background daemon
- No application integration required
- Requires “mount with discard” option
- Transparent Logical/Physical space management by the CSD

**IOCTL or SYSFS API**

**SFX Space Balancer**

*Actively monitor true free space via API or Transparently manage free space via Space Balancer*
# CSD 2000 – Space Balancer Example

## 1. Format Configuration
(6.4TB Format)

- **6.4TB Logical Capacity**: 5.4TB Free
- **3.2TB Physical Capacity**: 2.7TB Free
- **3.2TB Raw Capacity CSD 2000**: 0.5TB Data A

## 2. Data Set A Written
(1TB, 2:1 Compressible Data)

- **5.4TB Free**
- **1.0TB Data A**

## 3a. Data Set B Written
(1TB, Incompressible Data)

- **4.4TB Free**
- **1.0TB Data B**

## 3b. Space Balancer Intercedes

- **3.4TB Free**
- **1.0TB Balancer File**

### Projected Compression Ratio (PCR) set to 2:1; Balancer File = 0KB

<table>
<thead>
<tr>
<th>PCR</th>
<th>Free</th>
<th>Has the PCR been met?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:1</td>
<td>5.4/2.7</td>
<td>Yes</td>
</tr>
<tr>
<td>2:1</td>
<td>4.4/1.7</td>
<td>No</td>
</tr>
<tr>
<td>2:1</td>
<td>(4.4−1)/1.7</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**ScaleFlux**
Transparent Hardware Compression Will Be Pervasive

Most all-flash array (AFA) products today

Solid-state drive (SSD)

Cloud-native transparent compression

How relational database can take full advantage of transparent compression
A Simple Idea: Dual In-Memory vs. On-Storage Page Format

- **In-memory** row-based pages

  - 8KB page
    - $e_{1,1}, e_{1,2}, \ldots, e_{1,n}$
    - $e_{2,1}, e_{2,2}, \ldots, e_{2,n}$
    - $\cdots$
    - $e_{m,1}, e_{m,2}, \ldots, e_{m,n}$
    - 1st row
    - 2nd row
    - m-th row

- **On-disk** column-based pages

  - 8KB page
    - $f_1(e_{1,1}, e_{2,1}, \ldots, e_{m,1})$
    - $f_2(e_{1,2}, e_{2,2}, \ldots, e_{m,2})$
    - $\cdots$
    - $f_n(e_{1,n}, e_{2,n}, \ldots, e_{m,n})$
    - 1st column
    - 2nd column
    - n-th column

- **In-memory page**: Conventional row-based format ➔ keep query engine intact
- **On-storage page**: Column-based format with per-column transformation (e.g., shuffle, XOR) ➔ higher page data compressibility ➔ Lower storage cost
- Demonstrate the concept by adding ~600 LoC into InnoDB ➔ Improve data compression ratio by ~40% with few percentage TPS loss
Another Simple Idea: Virtual Full-Page-Write

- Goal: Reduce write IO traffic caused by the *full-page-write* in WAL

- Pad zeros to make the full-page 4KB-aligned without sacrificing storage cost
- Leverage “relink” filerange-clone in Linux filesystem (e.g., XFS, ZFS and Btrfs)

Realize full-page-write through filerange-clone

Eliminate the write IO traffic caused by full-page-write
Summary

- In-storage data path compression: The perfect match with Postgres
- Volume deployment today
  - The most advanced Computational Storage Drive
  - Immediate impact of compute AND storage I/O acceleration

Questions? Interested in PoC?

[Links]
- www.scaleflux.com
- info@scaleflux.com
- tong.zhang@scaleflux.com