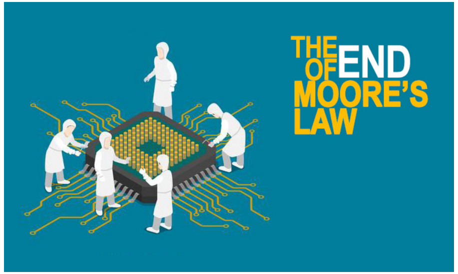
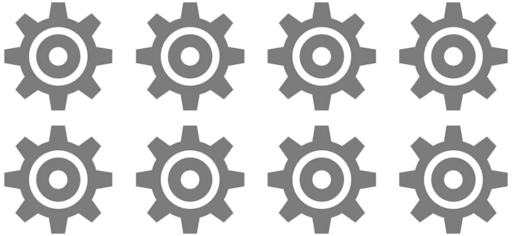
ScaleFlux®

Postgres Meets Computational Storage

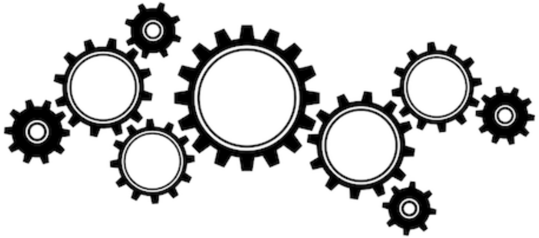
02/07/2021

The Rise of Computational Storage





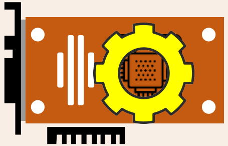
Homogeneous Computing



Heterogenous Computing

Compute

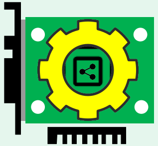
FPGA/GPU/TPU



End of Moore's Law

Networking

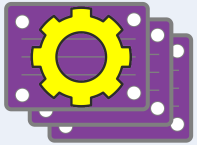
SmartNICs



10 → 100-400Gb/s

Storage

Computational Storage

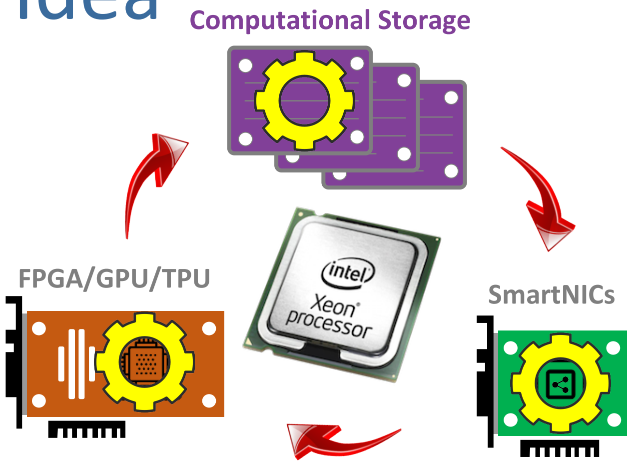


Fast & Big Data Growth

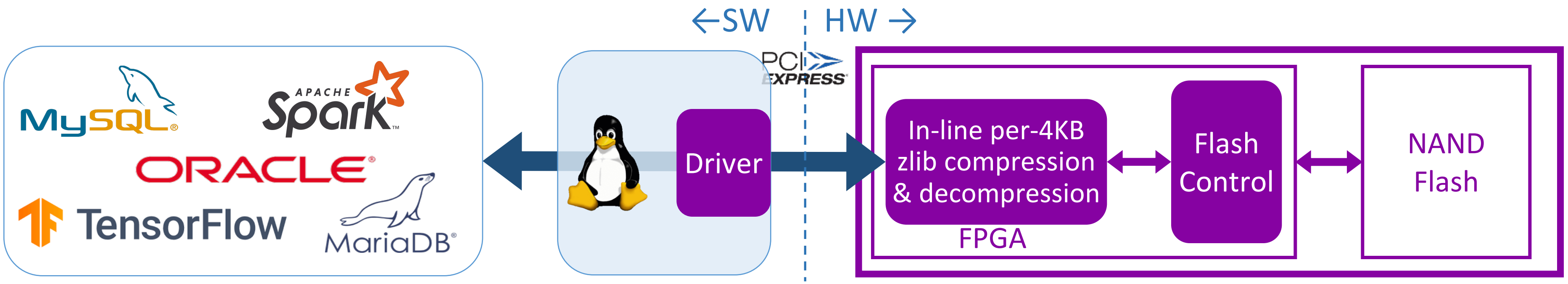
Domain Specific Compute

Computational Storage: A Simple Idea

- ❑ End of Moore's Law ➔ heterogeneous computing



Low-hanging fruits



Computational Storage Drive (CSD) with Data Path Transparent Compression

Data Compression

PostgreSQL

Filesystem w/ transparent compression (ZFS, Btrfs, …)

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

4KB-aligned

Wasted

Improve compression ratio & reduce storage space waste

Block size ↑ → R/W amplification ↑

lz4 → zlib/ZSTD → CPU overhead ↑

→ Performance ↓

Transparent Compression

PostgreSQL

Journaling filesystem (ext4, XFS, …)

Block layer w/ transparent compression (VDO, …)

4KB

A B C D E

Per-4KB lz4

Compression

4KB sector

A B | C | D E

Improve compression ratio &

reduce storage space waste

lz4 ➔ zlib/ZSTD ➔ CPU overhead ↑ ➔ Performance ↓

Transparent Compression

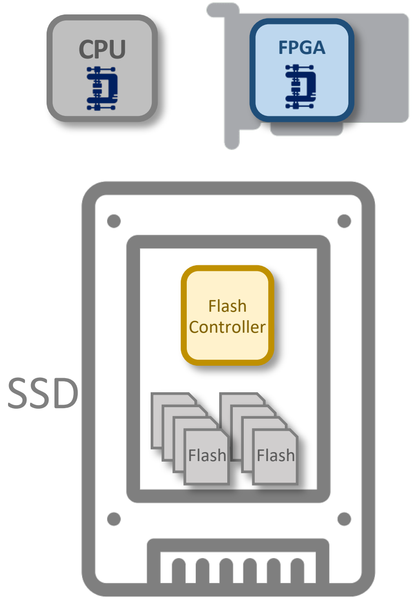
PostgreSQL

Journaling filesystem (ext4, XFS, …)

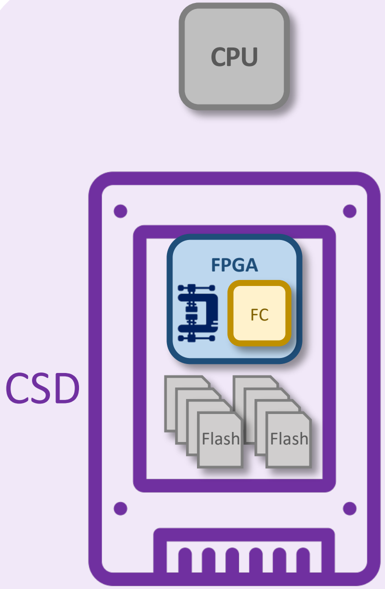
Normal block layer

Computational Storage Drive (CSD) with Data Path Transparent Compression

ScaleFlux Computational Storage Drive: CSD 2000



Multiple, discrete components
for Compute and SSD Functions



Single FPGA combines
Compute and SSD Functions

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

CSD 2000: Data Path Transparent Compression

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

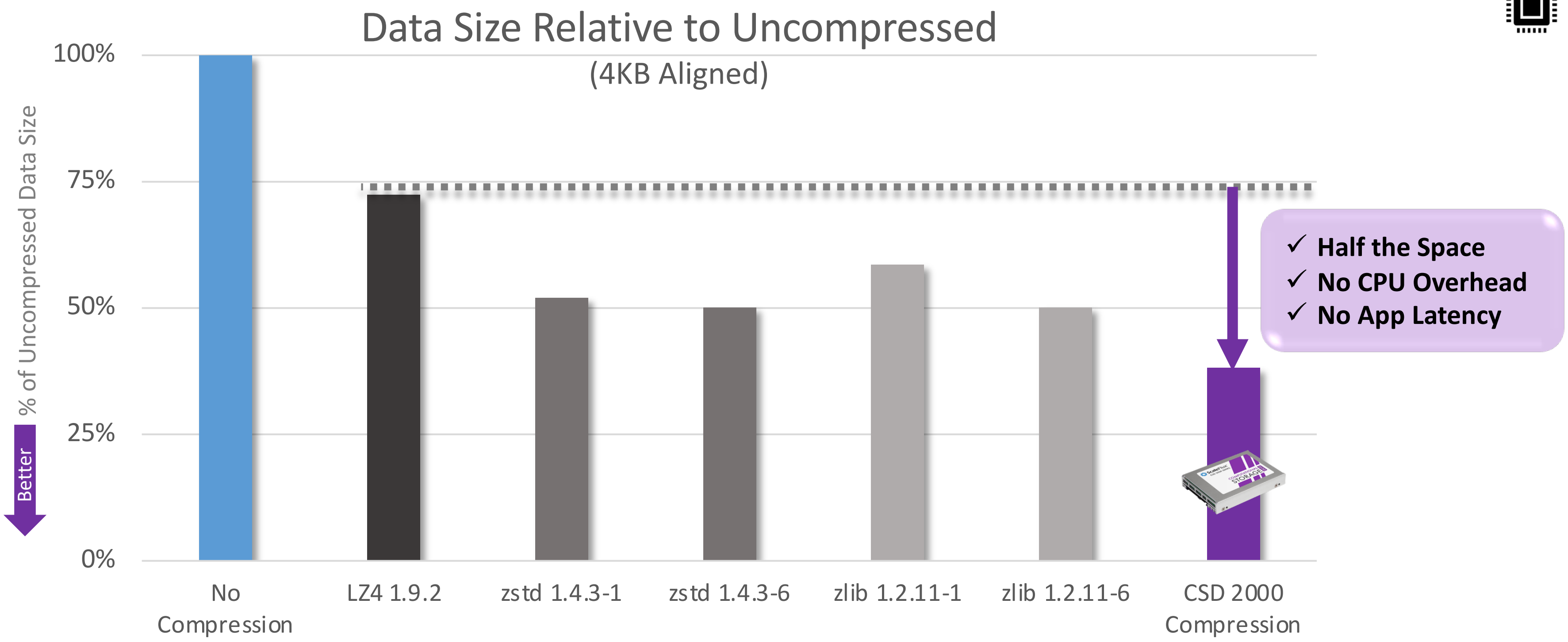
VS

ScaleFlux CSD 2000: Data path compression

HW zlib compression → Much higher compression ratio @ zero CPU cost

Tight packing → Zero space waste

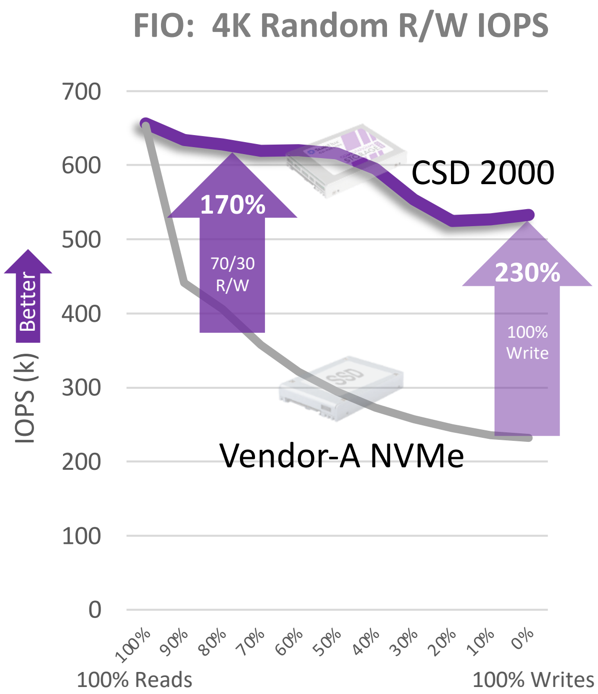
CSD 2000: Compression Without Compromise



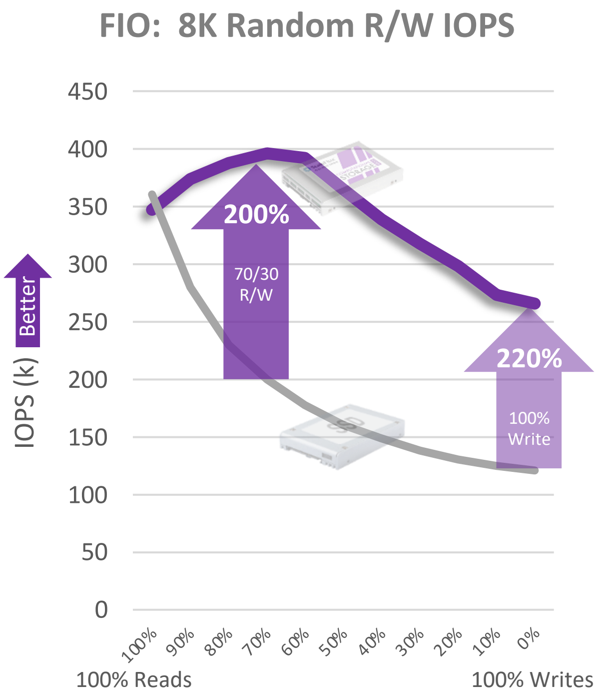
Achieve Greater Storage Space Savings with CSD 2000

Using Canterbury Corpus files (2,810,874B uncompressed); 8KB compression block to simulate relational databases (e.g. PostgreSQL, OracleDB, SQL Server)

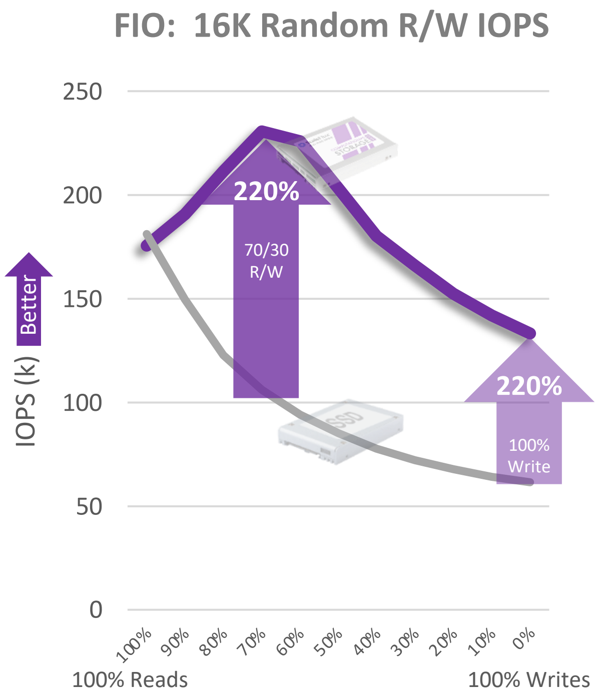
CSD 2000: Data Path Compression/Decompression



2.5:1 Compressible Data, 8 jobs, 32 QD, steady state after preconditioning



2.5:1 Compressible Data, 8 jobs, 32 QD, steady state after preconditioning



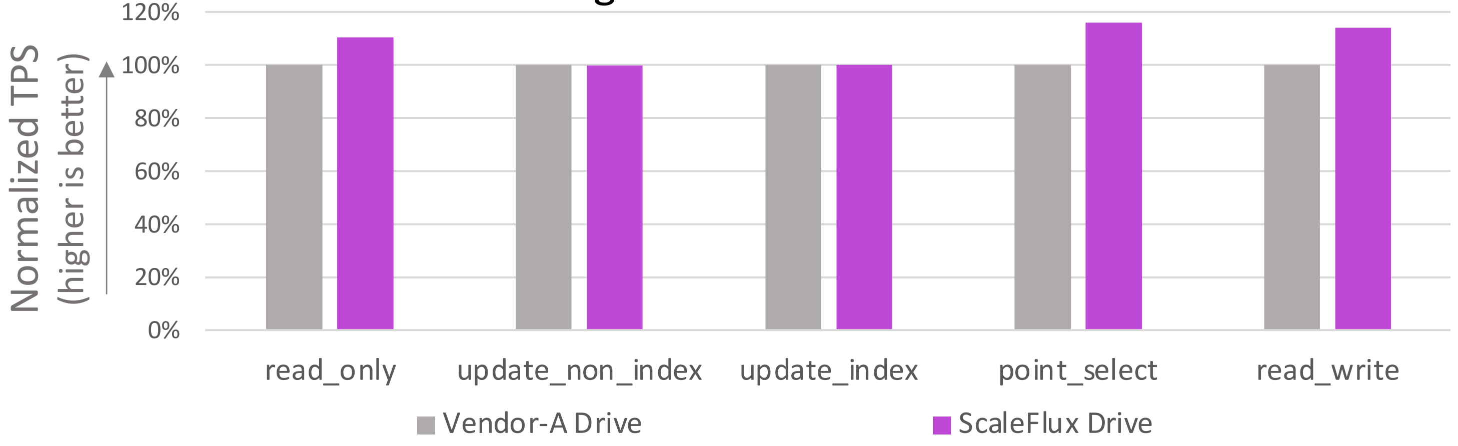
2.5:1 Compressible Data, 8 jobs, 32 QD, steady state after preconditioning

Increasing Mix R/W Performance Advantage with Larger Block Sizes

Straightforward Usage in Postgres

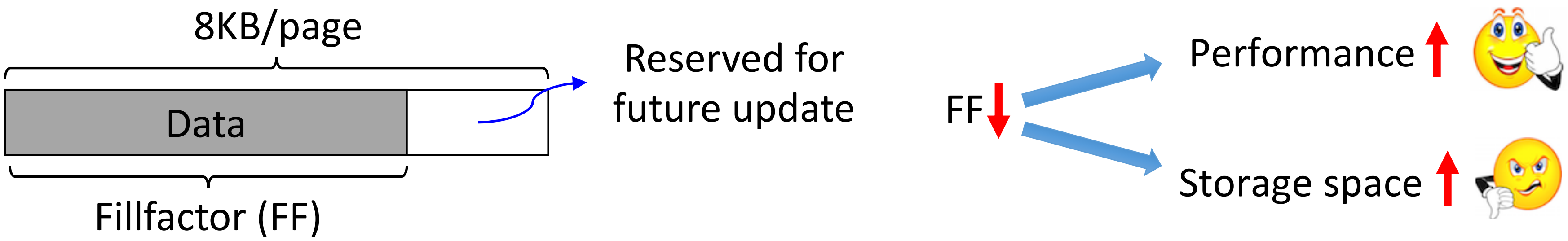
- Xeon E5-2667 v4 32-core @ 3.2GHz, 256GB DRAM
- CentOS 7.5.1804, Postgres 10.10, Sysbench 1.1.0 (64 threads)
- 3.2TB vendor-A NVMe drive vs. 3.2TB ScaleFlux drive
- Transparent compression: **2TB** Postgres dataset ➔ **776GB** (60% reduction)

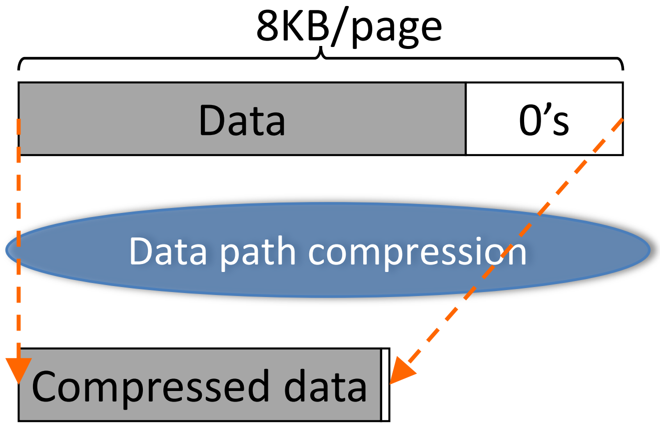
Where does the TPS gain on read-intensive workloads come from?

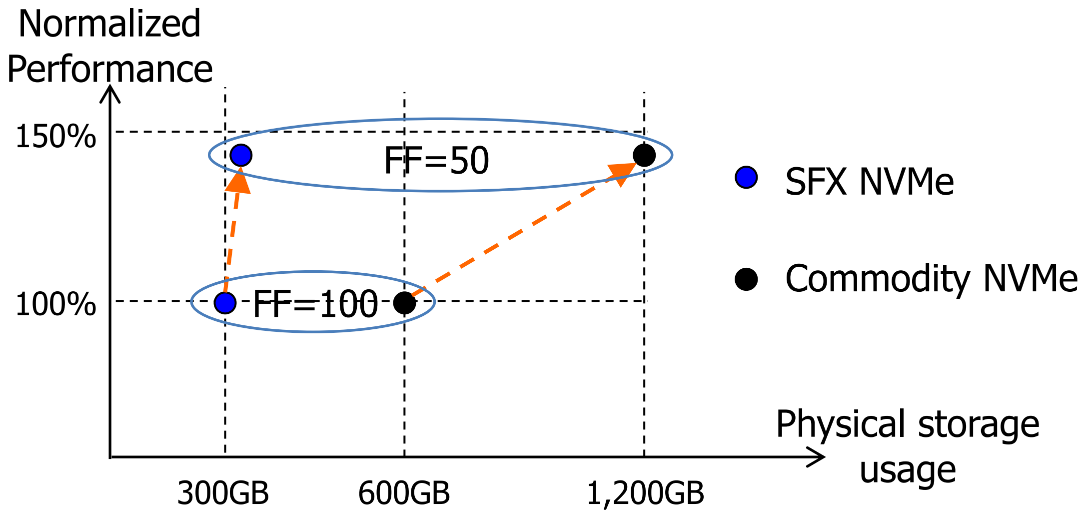


Smaller storage footprint
↓
Less flash memory access conflict
↓
Higher page read throughput

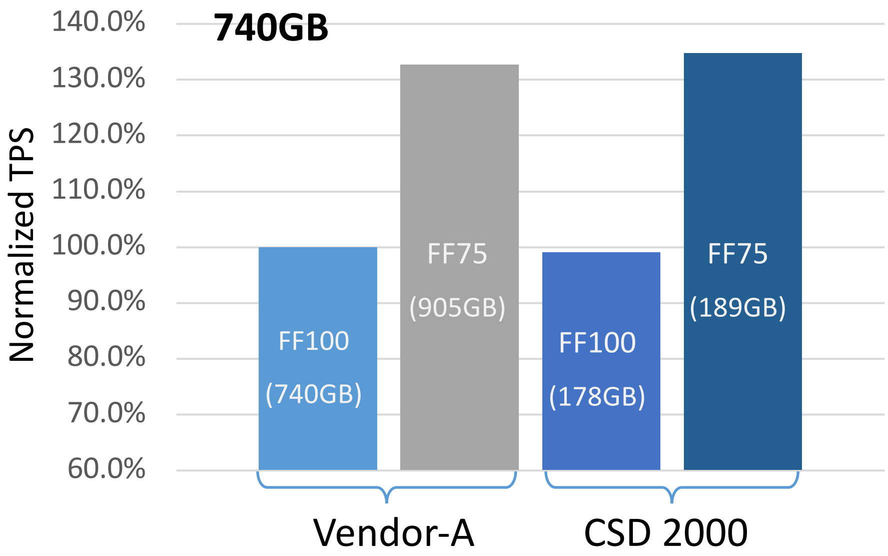
One Step Further

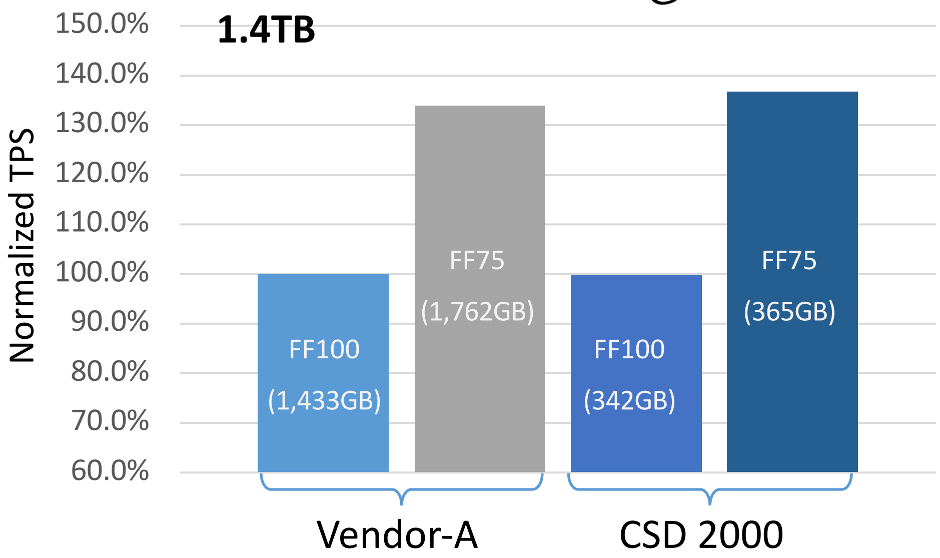






One Step Further (Sysbench-TPCC)



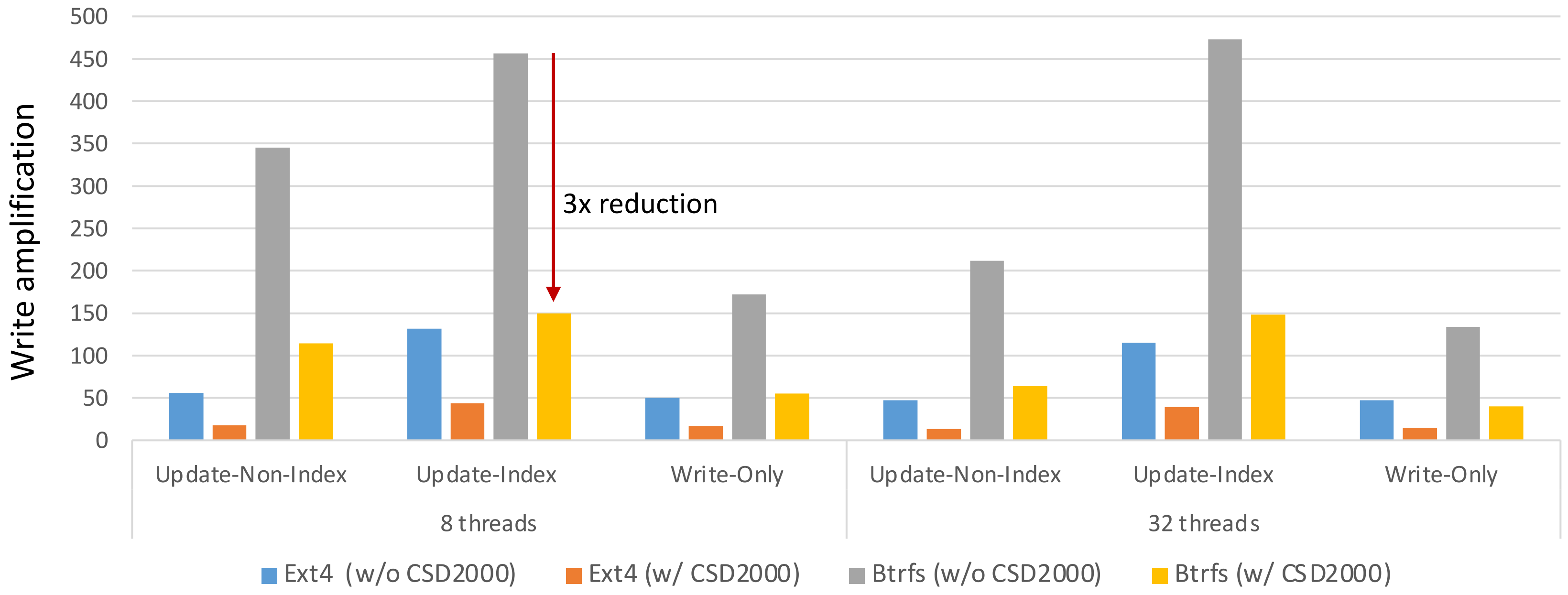


Fillfactor	Drive	Logical size (GB)	Physical size (GB)	Comp Ratio
100	Vendor-A	740	740	1.00
	CSD 2000		**178**	4.12
75	Vendor-A	905	905	1.00
	CSD 2000		**189**	4.75

Fillfactor	Drive	Logical size (GB)	Physical size (GB)	Comp Ratio
100	Vendor-A	1,433	1,433	1.00
	CSD 2000		**342**	4.19
75	Vendor-A	1,762	1,762	1.00
	CSD 2000		**365**	4.82

Write Amplification

- 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

IOCTL or SYSFS API

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

Ext4 File System

XFS Filesystem

ORACLE

AEROSPIKE

Automatic Storage Management (ASM)

SFX Space Balancer

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

Ext4 File System

XFS Filesystem

Actively monitor true free space via API or Transparently manage free space via Space Balancer

CSD 2000 – Space Balancer Example

3.2TB Raw Capacity CSD 2000

1. Format Configuration
(6.4TB Format)

6.4TB Logical Capacity

3.2TB Physical Capacity

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

$$\frac{Free\ C}{Free\ PC} = PCR$$

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

5.4TB Free

1. 0TB Data A

2.7TB Free

0.5TB Data A

Has the PCR been met?

$\frac{5.4}{2.7} \le 2?$ ✓ Yes

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

4.4TB Free

1. 0TB Data B

1. 0TB Data A

1.7TB Free

1. 0TB Data B

0.5TB Data A

Has the PCR been met?

$\frac{4.4}{1.7} \le 2?$ ✘ No

3b. Space Balancer Intercedes

3.4TB Free

1.0TB Balancer File

1. 0TB Data B

1. 0TB Data A

1.7TB Free

1. 0TB Data B

0.5TB Data A

Has the PCR been met?

$\frac{(4.4-1)}{1.7} \le 2?$ ✓ Yes

Transparent Hardware Compression Will Be Pervasive

Most all-flash array (AFA) products today

PowerMax
DELL EMC

Hewlett Packard Enterprise
nimblestorage

PURESTORAGE

Solid-state drive (SSD)

ScaleFlux™

SEAGATE

Many more on their way

Cloud-native transparent compression

amazon web services
Graviton2 processor

Microsoft Azure
Corsia Processor

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 ⟷ Row-column conv./trans. ⟷ **On-disk** column-based pages

8KB page

$e_{1,1}, e_{1,2}, \ldots, e_{1,n}$	$e_{2,1}, e_{2,2}, \ldots, e_{2,n}$	· · ·	$e_{m,1}, e_{m,2}, \ldots, e_{m,n}$
1st row	2nd row	⋮	m-th row

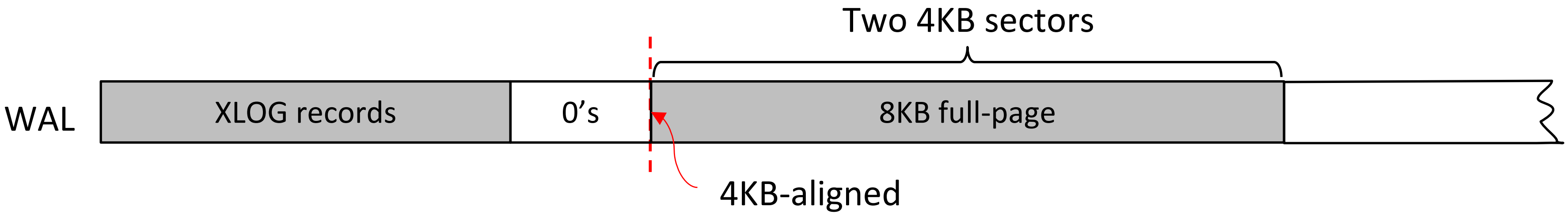
8KB page

$f_1(e_{1,1}, e_{2,1}, \ldots, e_{m,1})$	$f_2(e_{1,2}, e_{2,2}, \ldots, e_{m,2})$	· · ·	$f_n(e_{1,n}, e_{2,n}, \ldots, e_{m,n})$
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 "reflink" 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?

www.scaleflux.com

info@scaleflux.com

tong.zhang@scaleflux.com

ScaleFlux™