How Can Database Capitalize on Computational Storage

JB Baker and Tong Zhang, ScaleFlux

Agenda

- ScaleFlux Intro
- Computational Storage Intro
- Benefits of Computational Storage for Databases
- Examples of Achieving the Benefits

ScaleFlux: Computational Storage Leader

Low Latency Flash Storage

Compute Engines

Industry Standard Infrastructure

Data Driven Applications

Database | Big Data | AI/ML | IOT/Edge | CDN

- Significant NVM storage & algorithm expertise
- **1st Production Computational Storage**
- **2nd Generation of CSD shipping**
- US & China manufacturing sites
- Hyperscale, webscale & enterprise customers

FMS18 Best of Show
Most Innovative Startup

FMS19 Best of Show
Most Innovative Flash Memory Customer Implementation
+ Alibaba

IDC Innovator 2019
https://www.idc.com/getdoc.jsp?containerId=US45416319

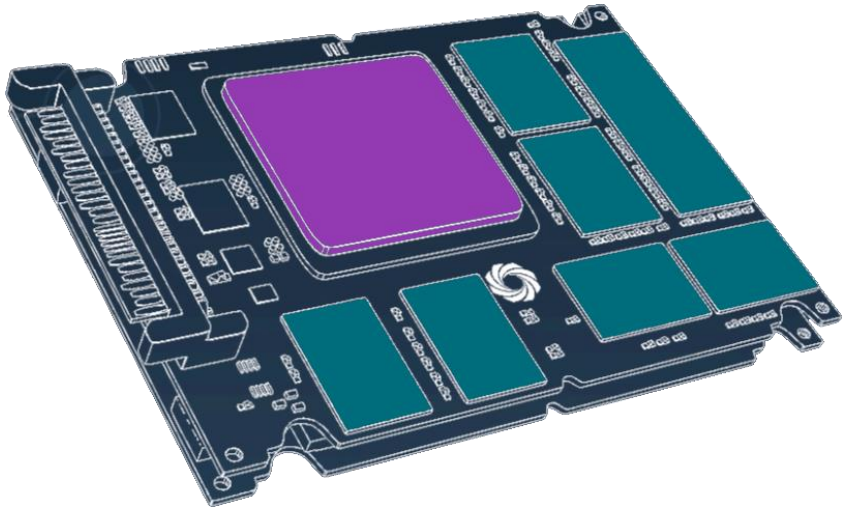
What is Computational Storage?
(and more importantly Why?)

What is Computational Storage?

- Storage Networking Industry Association (www.snia.org)
  - Architectures that provide Computational Storage Functions coupled to storage, *offloading host processing or reducing data movement*.

  - These architectures enable improvements in application performance and/or infrastructure efficiency through the integration of compute resources (outside of the traditional compute & memory architecture) either directly with storage or between the host and the storage. The goal of these architectures is to *enable parallel computation and/or to alleviate constraints on existing compute, memory, storage, and I/O*

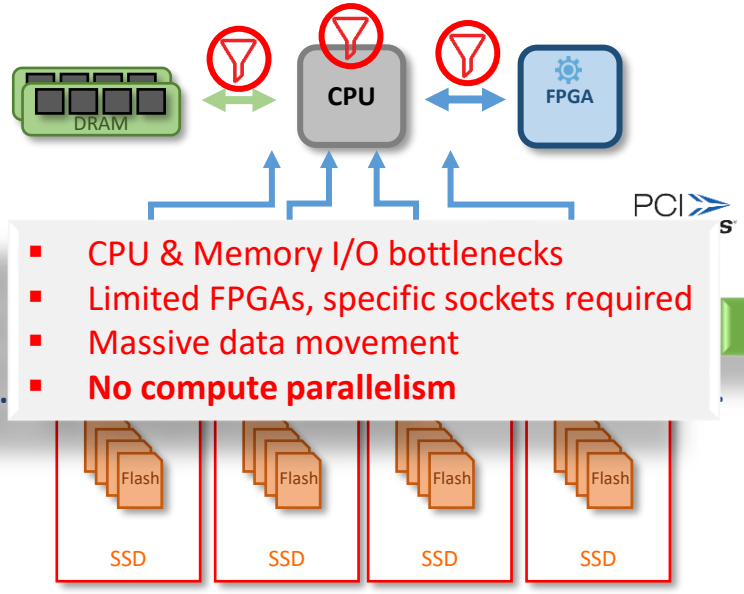
What is a Computational Storage Drive (CSD)?



Enterprise PCIe SSD + Compute Offload Engines

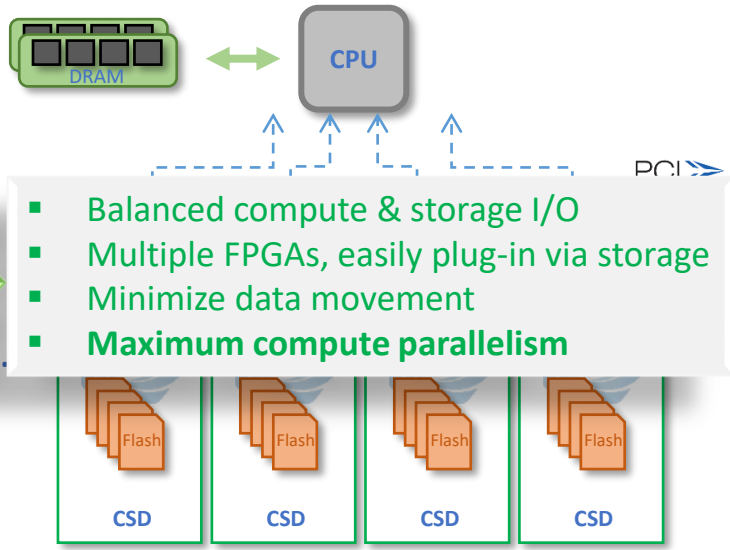
Why use Computational Storage?

Processor-Driven Architecture



- CPU & Memory I/O bottlenecks
- Limited FPGAs, specific sockets required
- Massive data movement
- **No compute parallelism**

Data-Driven Architecture



- Balanced compute & storage I/O
- Multiple FPGAs, easily plug-in via storage
- Minimize data movement
- **Maximum compute parallelism**

CSD: Computational Storage Drive

Advantages of Computational Storage for Databases
(and how to realize them)

Benefits of Computational Storage with Databases

Increase Performance

- Increase QPS / TPS
- Reduce Latency
- Alleviate bottlenecks
  - Processors
  - Storage Buses
  - Network

Reduce Costs

- Total infrastructure
- Power
- Cost per
  - Query
  - Transaction
  - GB of Storage

MySQL® MariaDB® PostgreSQL

Realizing the Benefits of CSDs for Databases…

Without Code Changes

- Transparent Functions

- Today:
  - ***Compression/Decompression***
- Future:
  - Encryption/Decryption

Through Minor Tweaks

- *Non*-transparent Functions
- *Modify existing* storage engines
- *Modify existing* DB

- Today:
  - Predicate Filtering
- Future:
  - ***Sparse logging***
  - ***Dual-Mode Page Format***

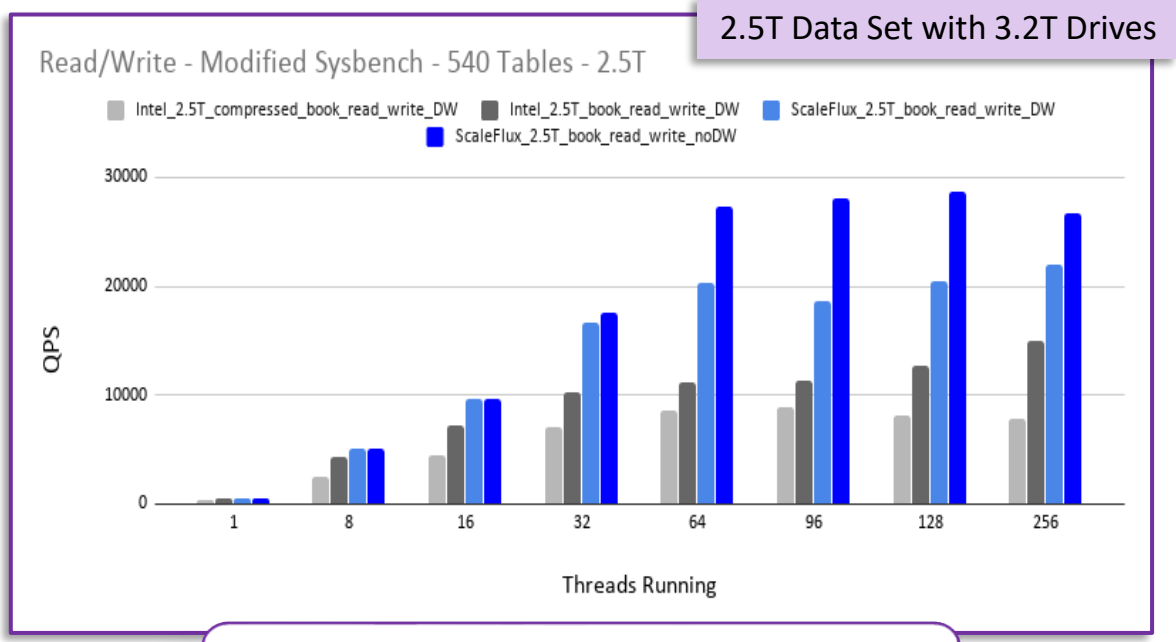
Through Innovation

- Non-transparent Functions
- *New Data Structures*
- *Optimized storage engine*

- Today:
  - n/a
- Future:
  - ***Sparse Data Structures (KV)***

Transparent Compression: Performance

2.5T Data Set with 3.2T Drives

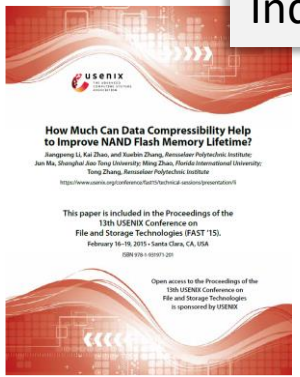


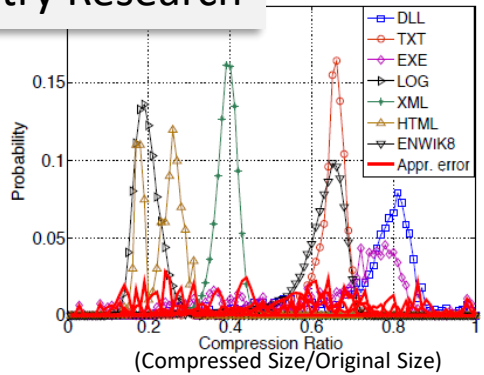
- Up to ***2x QPS*** vs NVMe with DWB On
- Up to ***2.7x QPS*** vs NVMe with DBW Off
- Garbage Collection reduces NVMe performance

CSD 2000 with Transparent Compression improves QPS

Transparent Compression: Cost

Industry Research





Customer Feedback

File Types	Data Compressibility (Original Size:Compressed Size)
Images, Video, Encrypted	<1.2:1
Binaries, DLL, EXE	1.2:1
XML	2:1
HTML, Logs, Database	**>2:1**

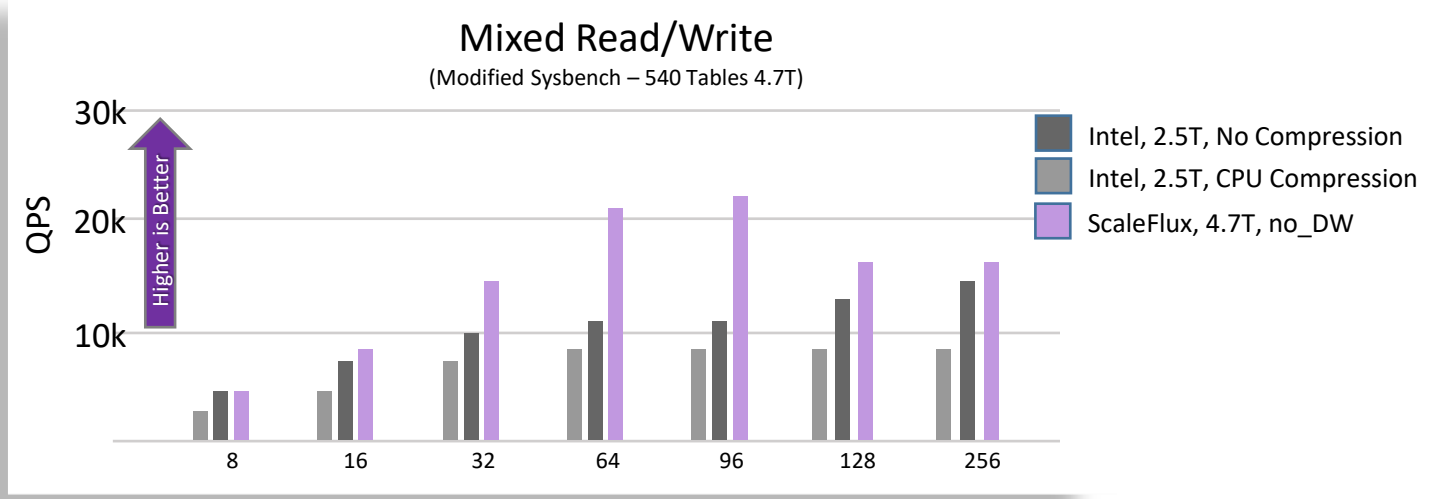
Customer type	Compression Ratio with CSD 2000 (Original Size:Compressed Size)
Hyperscale	**3:1**
Hyperscale	**4:1**
Webscale	**2.5:1**
CSP	**5:1**

Cut MySQL Storage Footprint by 50-80% with Compression

https://www.usenix.org/system/files/conference/fast15/fast15-paper-li.pdf

Transparent Compression:

Combining Cost Savings with Performance Enhancements



CPU Compression Penalizes Performance

- Up to 50% ***worse*** vs NVMe *without* Compression
- 1.6:1 compression ratio achieved

CSD 2000 Compression Improves Performance

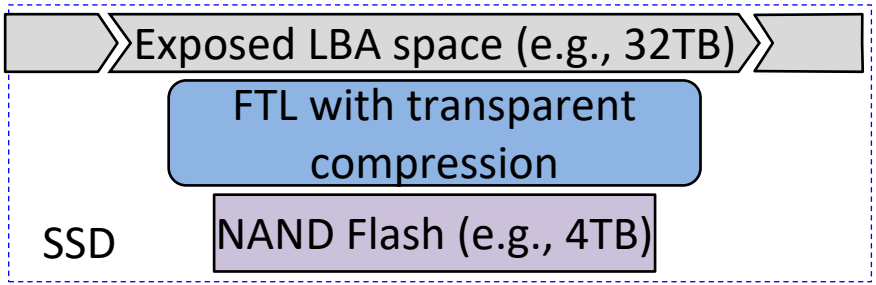
- ***2x QPS*** vs NVMe with CPU compression
- 1.8:1 compression ratio achieved

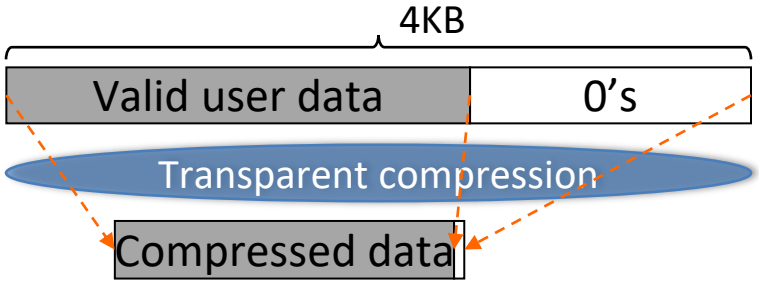
Source: https://www.scaleflux.com/downloads/Testing-the-Value-of-ScaleFlux-200806.pdf

Open the Door for System Optimization and Innovation

Logical storage space utilization efficiency

Physical storage space utilization efficiency



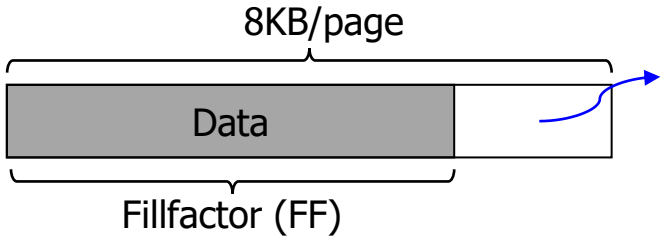


Unnecessary to use all the LBAs

Unnecessary to fill each 4KB sector with user data

OS/Applications can ***purposely waste*** logical storage space to gain benefits

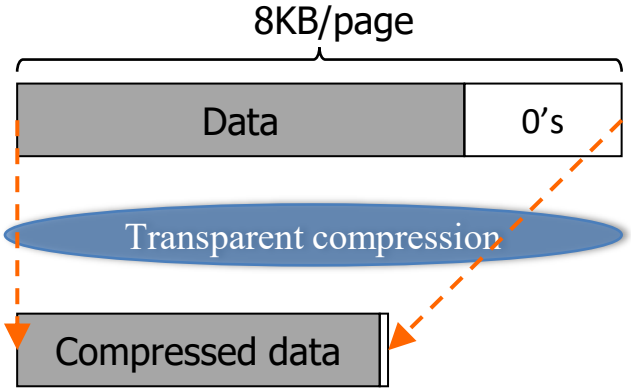
Optimization: Cost & Performance

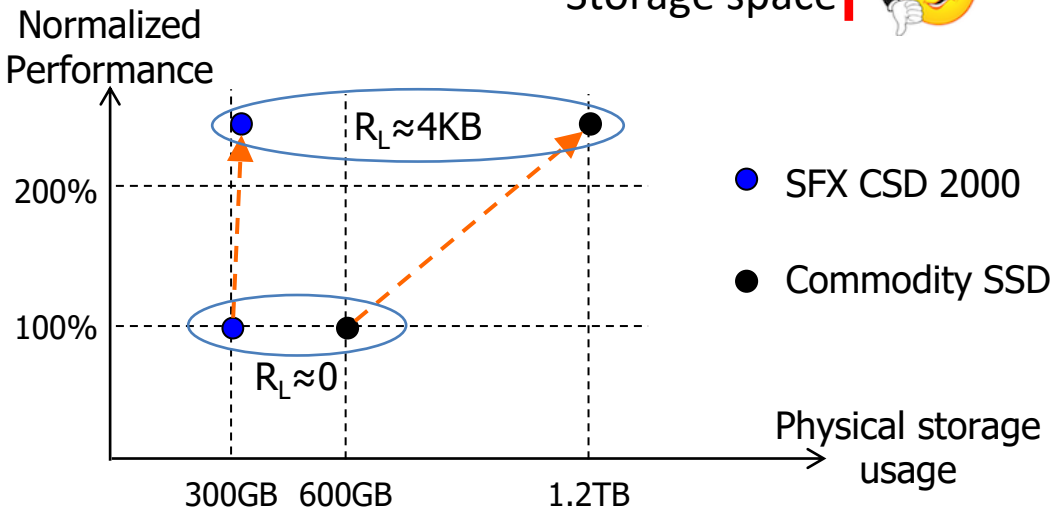


Reserved for future update

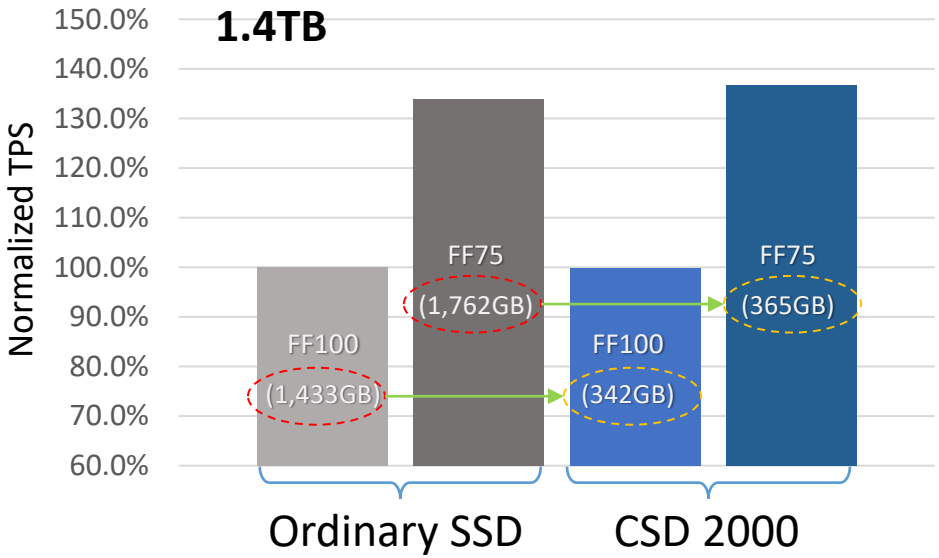
FF ↓ → Performance ↑

FF ↓ → Storage space ↑





Optimization: Cost & Performance



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

Performance

- *Similar at a given Fill Factor*
- *>30% higher at FF75 vs FF100*

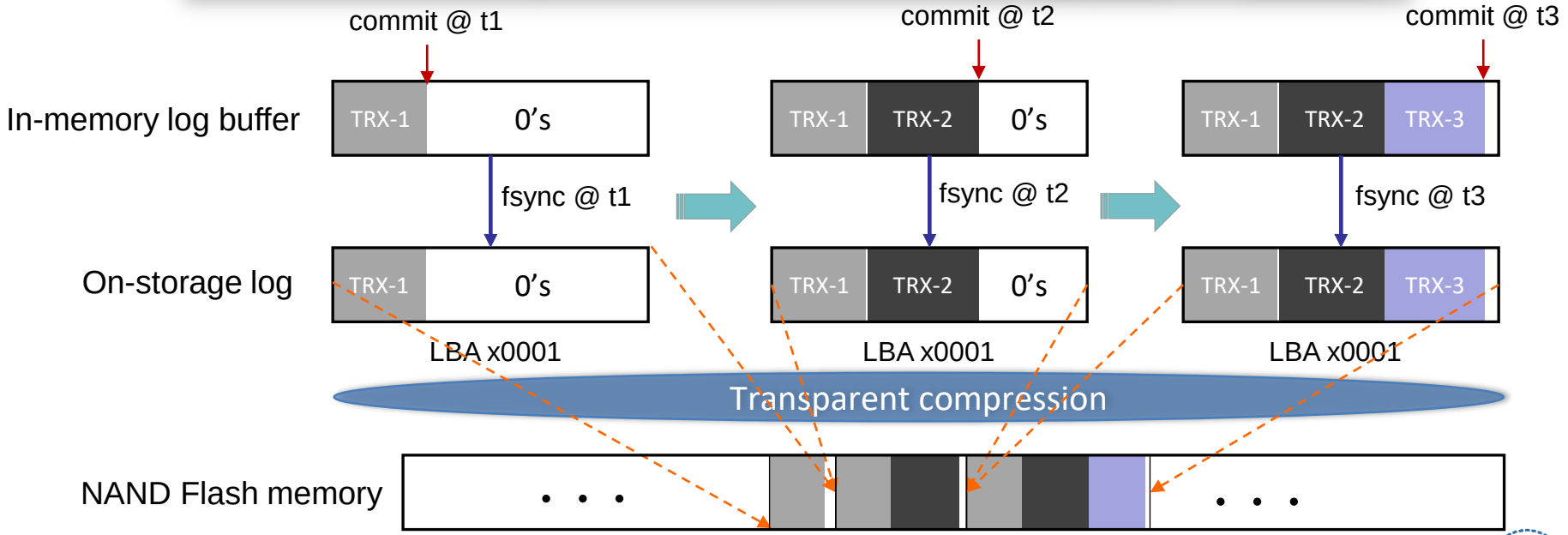
Cost

- *Data Footprint without Compression*
  - *+23% at FF75 vs FF100*
- *Data Footprint with Compression:*
  - *-76% at FF100*
  - *-79% at FF75*

Minor Tweaks: Redo/Binary Log

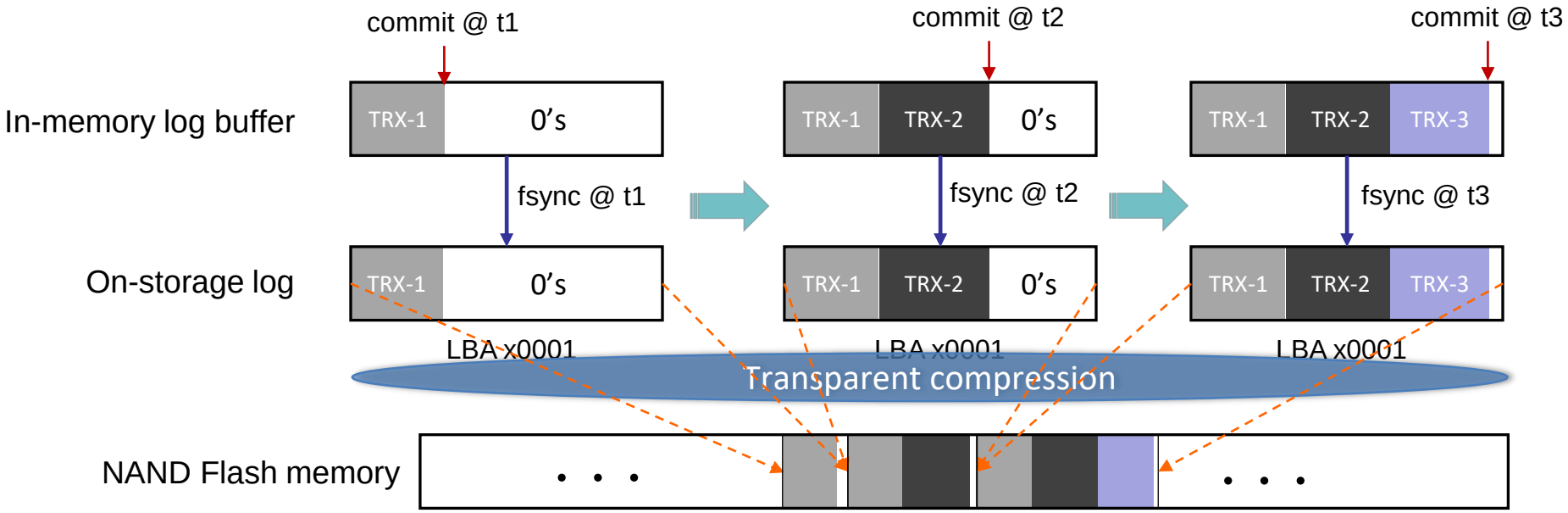
Current Problem

InnoDB redo/binary log: A critical component to ensure *atomicity* and *durability*



Minor Tweaks: Redo/Binary Log

Current Problem



Write amplification

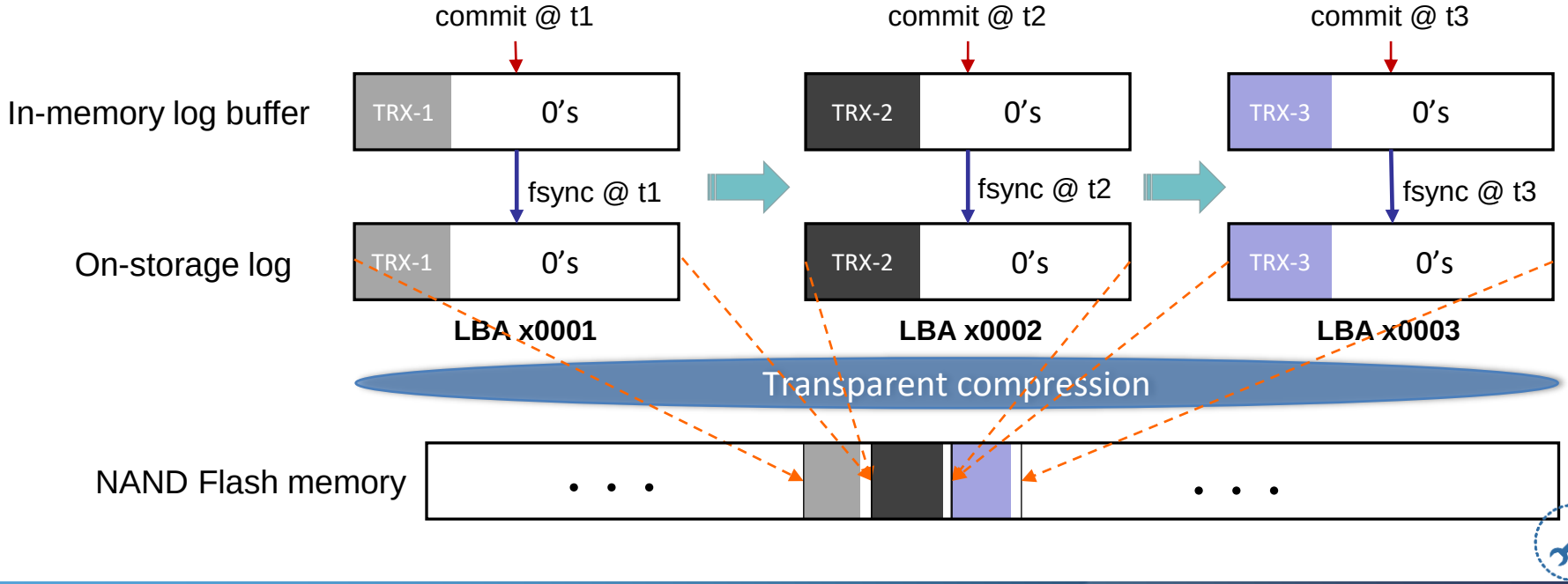
- More interference with other IOs
- Shorter NAND flash memory lifetime

Minor Tweaks: ***Sparse*** Redo/Binary Log

How it works

Sparse redo/binary log: Allocate a new 4KB sector per transaction commit

Waste logical storage space ➔ reduce logging-induced write amplification

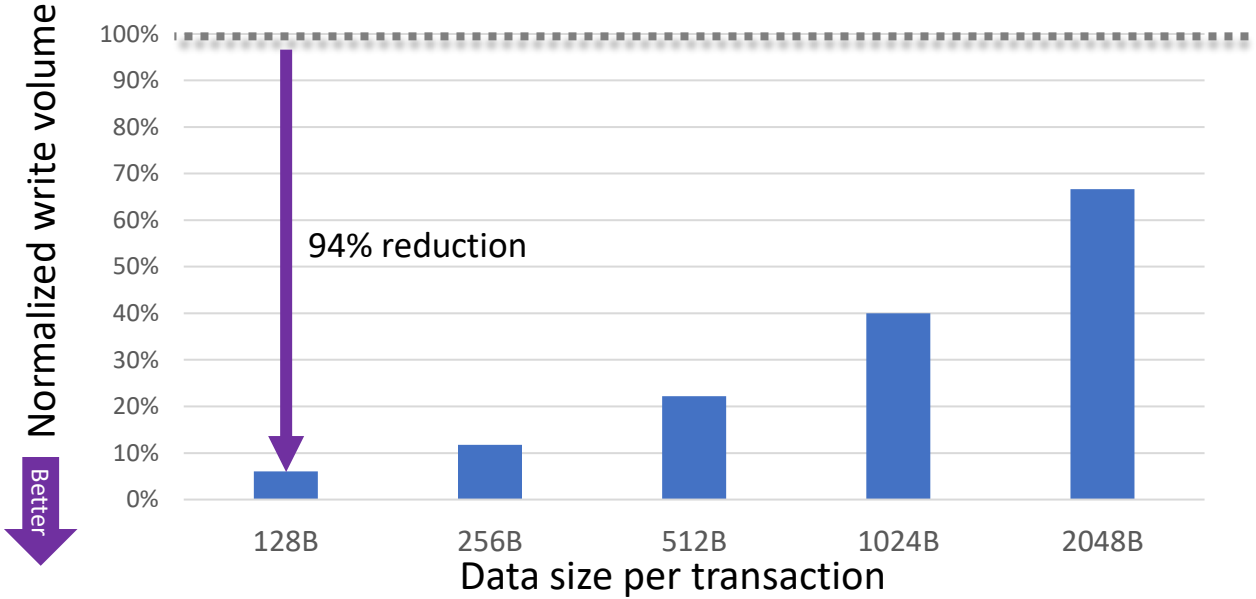


Minor Tweaks: Sparse Redo/Binary Log

Cost & Performance

Sparse redo/binary log: Allocate a new 4KB sector per transaction commit

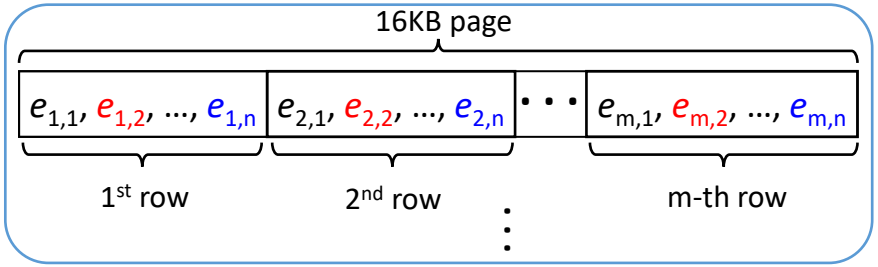
Waste logical storage space ➔ reduce logging-induced write amplification

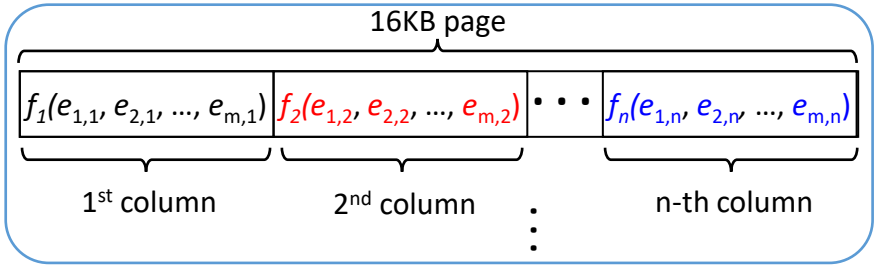


Minor Tweaks: Dual-Mode Page Format

Cost

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

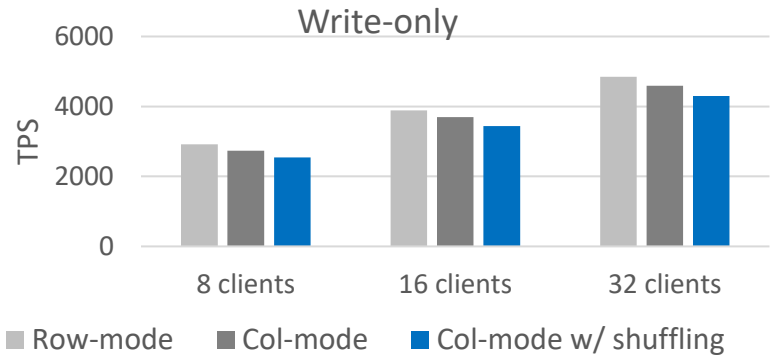


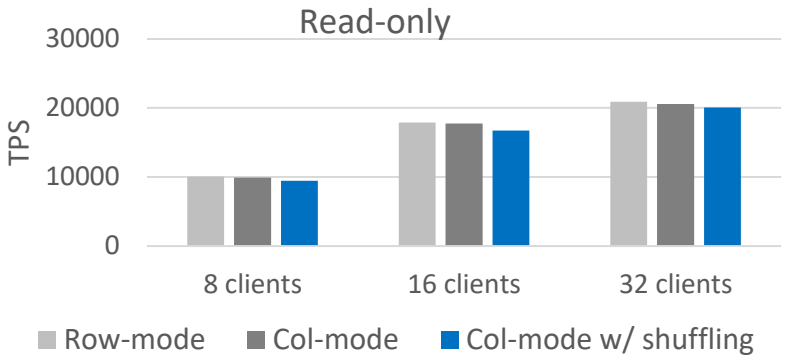


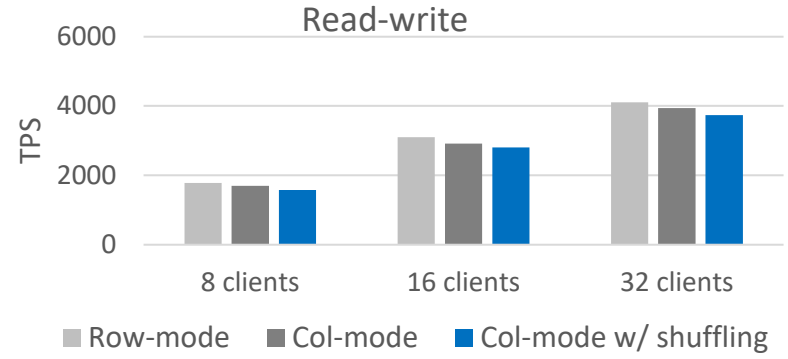
- **In-memory page**: Conventional row-based format ➔ keep query engine intact
- **On-storage page**: Column-based format with per-column transformation (e.g., shuffle, XOR) ➔ higher page data compressibility ➔ ***Lower storage cost***
- Demonstrate the concept by adding ~600 LoC into InnoDB

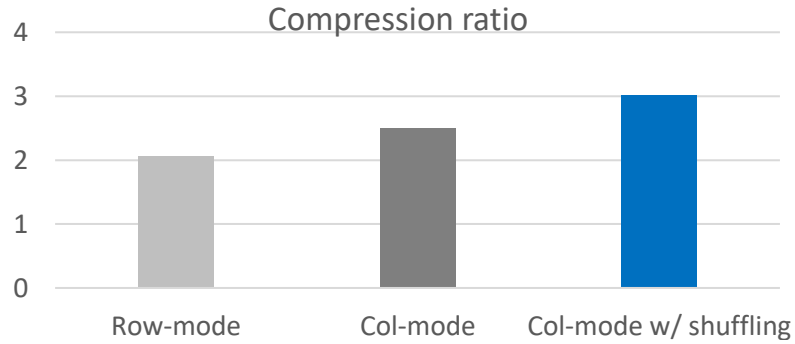
Minor Tweaks: Dual-Mode Page Format

Sysbench Results







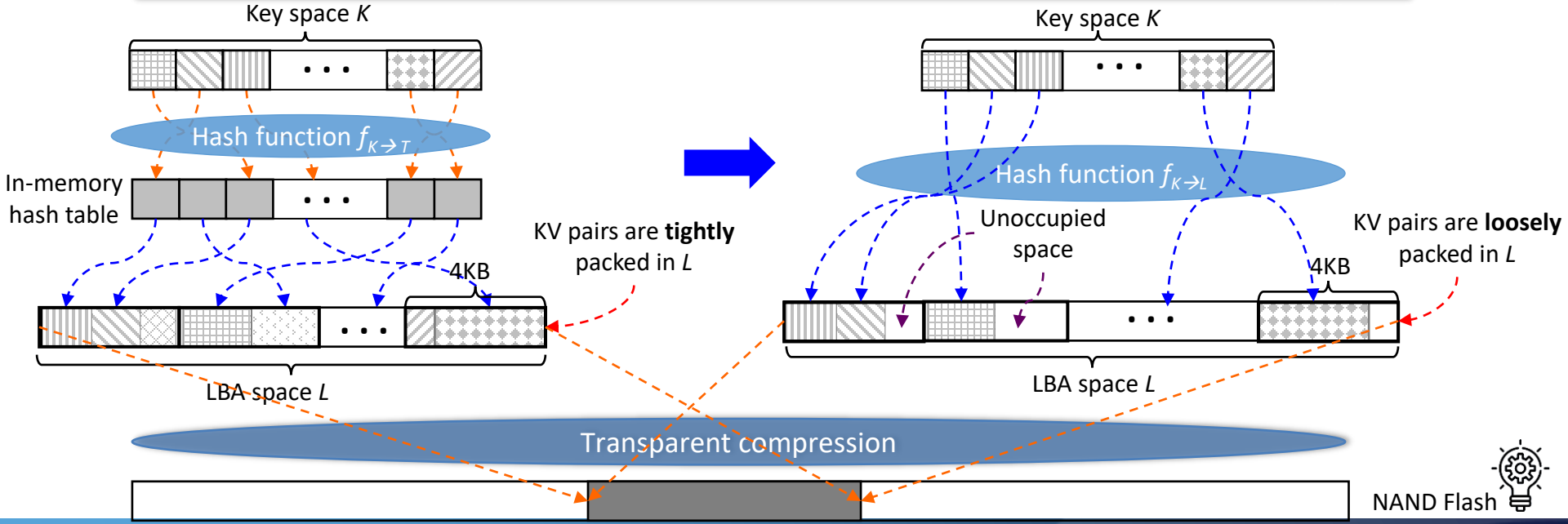


Row-column page conversion with shuffling reduces storage cost by 50% at few % TPS loss

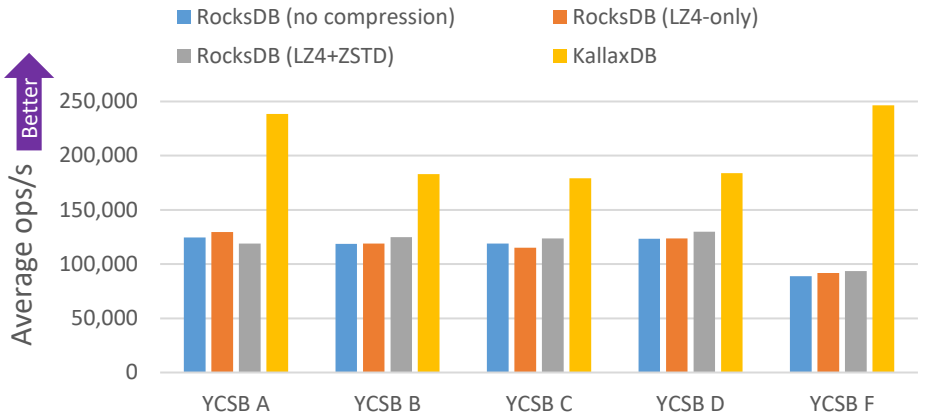
Innovation: Sparse Data Structures

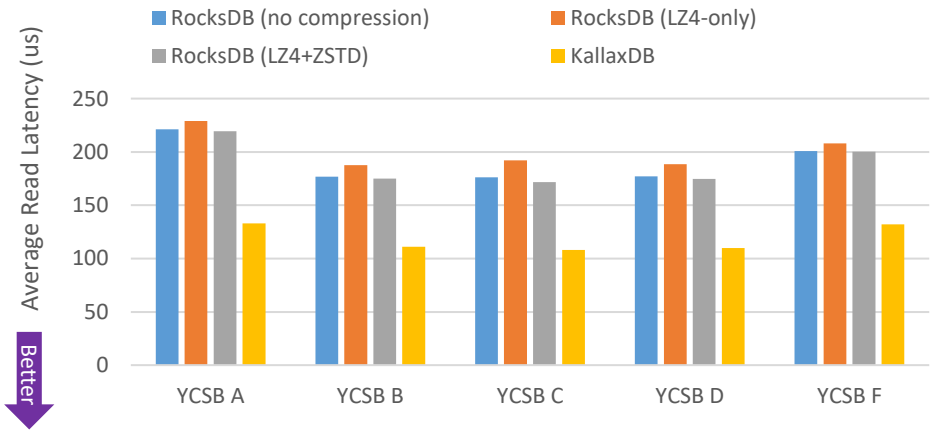
Table-less Hash-based KV Store

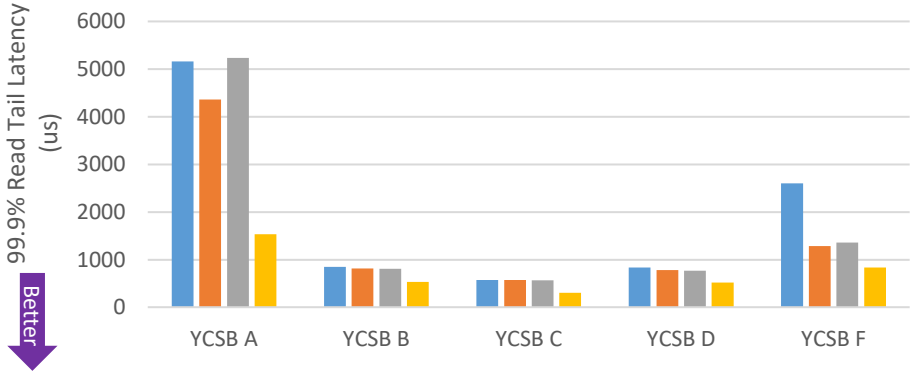
- Very simple idea
  - Hash *key space* directly onto *logical storage space* ➔ eliminate the in-memory hash table
  - Transparent compression eliminates the "unoccupied space" from physical storage space

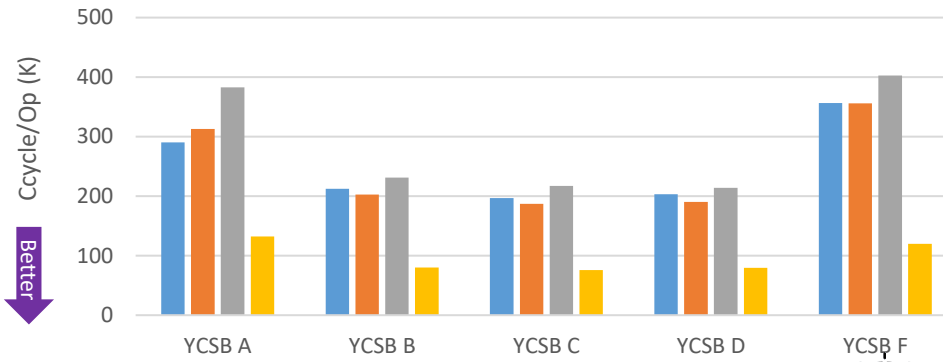


Innovation: Experimental Results (24 clients)









Summary

- **MariaDB, MySQL, PostgreSQL meet Computational Storage Drive**
  - Transparent compression & atomic write (w/o source code change)
  - Sparse log & dual-mode page format (w/ small source code changes)
- **CSD 2000 in volume deployment today**
  - Immediate benefits from compute AND storage I/O acceleration for performance & cost
- **Future enhancements**
  - Sparse data structures & optimized KV Store (Innovations)
- **More details & videos: www.scaleflux.com**

info@scaleflux.com tong.zhang@scaleflux.com jb.baker@scaleflux.com

ScaleFlux™

Thank You

97 East Brokaw Road, Suite 260
San Jose, CA 95112
www.scaleflux.com #compute2data

Sparse Data Structures: Table-less Hash-based KV Store

- Experimental Setup
  - 24-core 2.6GHz Intel CPU, 32GB DDR4 DRAM, and a 3.2TB SFX CSD2000
  - RocksDB 6.10 (12 compaction threads and 4 flush threads)
  - 400-byte KV pair size, 1 billion KVs ➔ 400GB raw data
  - Memory usage: RocksDB (5GB), KallaxDB (600MB)

YCSB A	50% reads, 50% updates
YCSB B	95% reads, 5% updates
YCSB C	100% reads
YCSB D	95% reads, 5% inserts
YCSB F	50% reads, 50% read-modify-writes

	Storage Usage
RocksDB (no compression)	428GB
RocksDB (LZ4-only)	235GB
RocksDB (LZ4+ZSTD)	201GB
KallaxDB	216GB