Performance Interference on Key-Value Stores in Multi-tenant Environments: When Block Size and Write Requests Matter

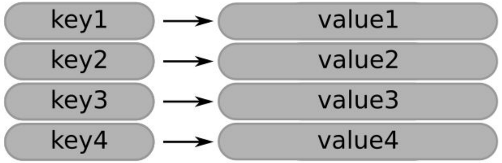
Adriano Lange, Tiago Rodrigo Kepe, Marcos Sfair Sunyé

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ
INSTITUTO FEDERAL
Paraná

LTB @ ICPE 2021

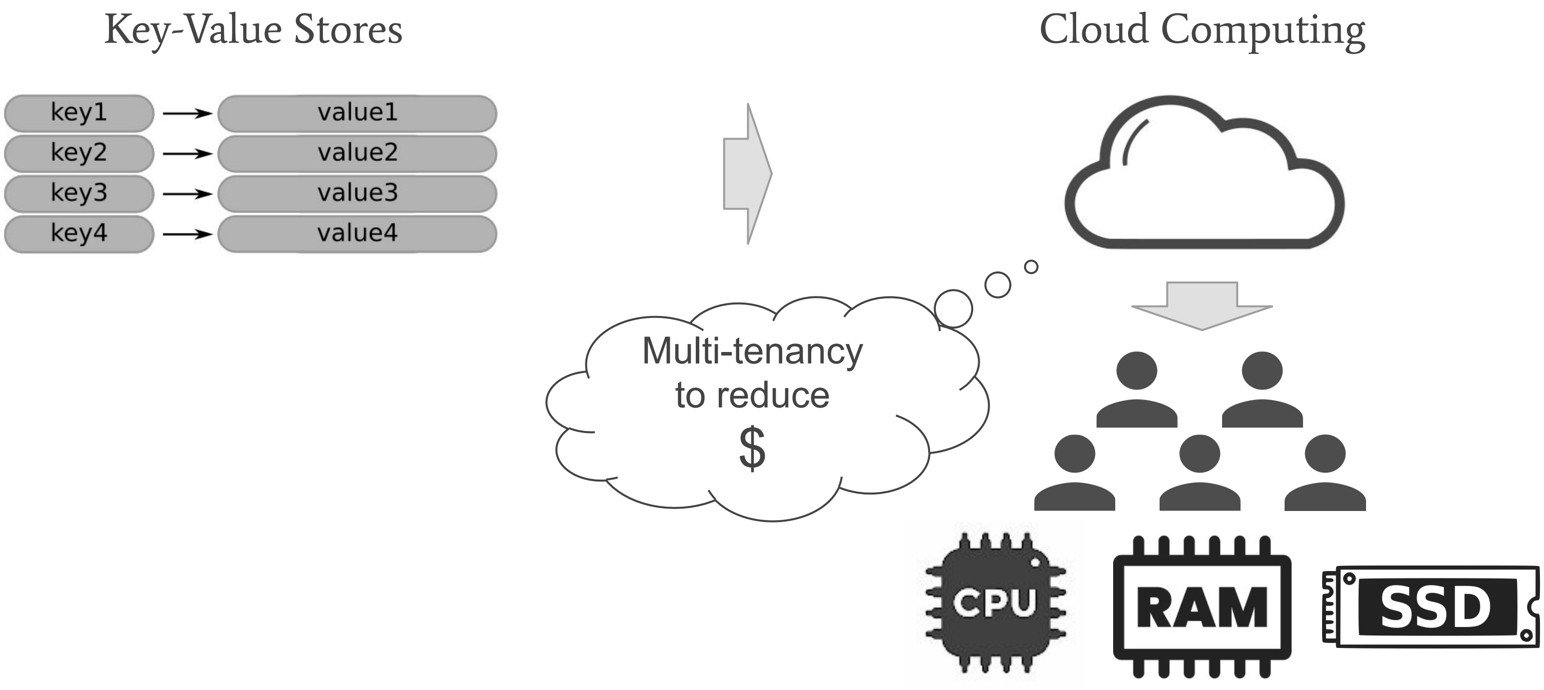
Key-Value Stores and Cloud Computing

Key-Value Stores



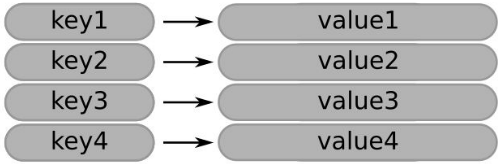
Cloud Computing

Key-Value Stores and Cloud Computing

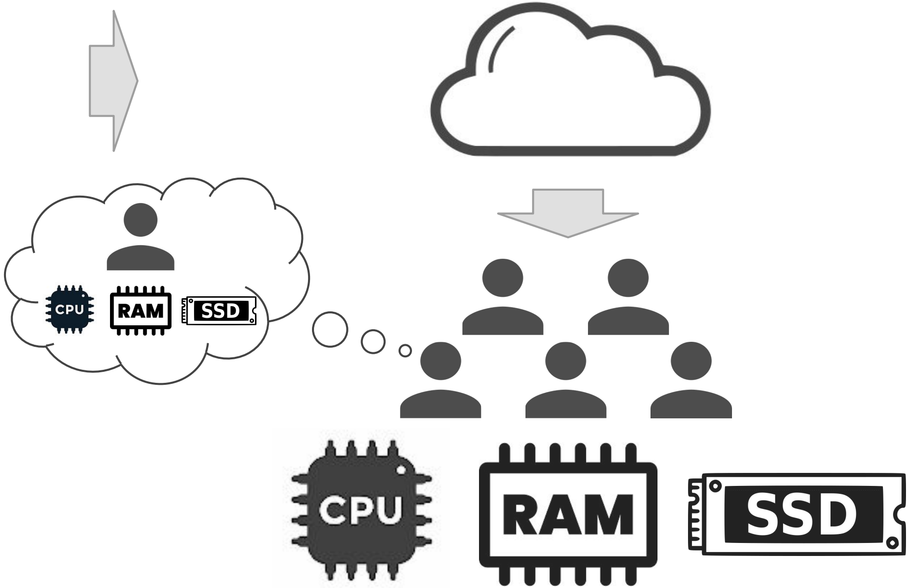


Key-Value Stores and Cloud Computing

Key-Value Stores



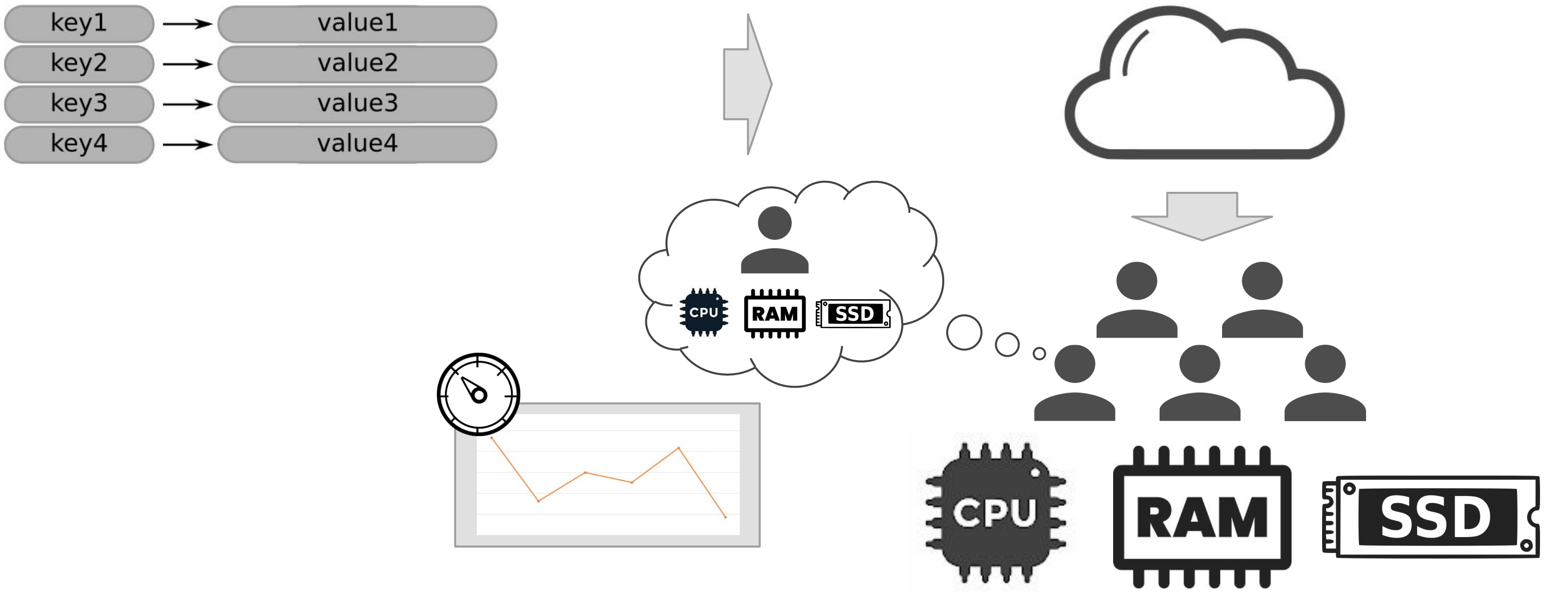
Cloud Computing



Key-Value Stores and Cloud Computing

Key-Value Stores

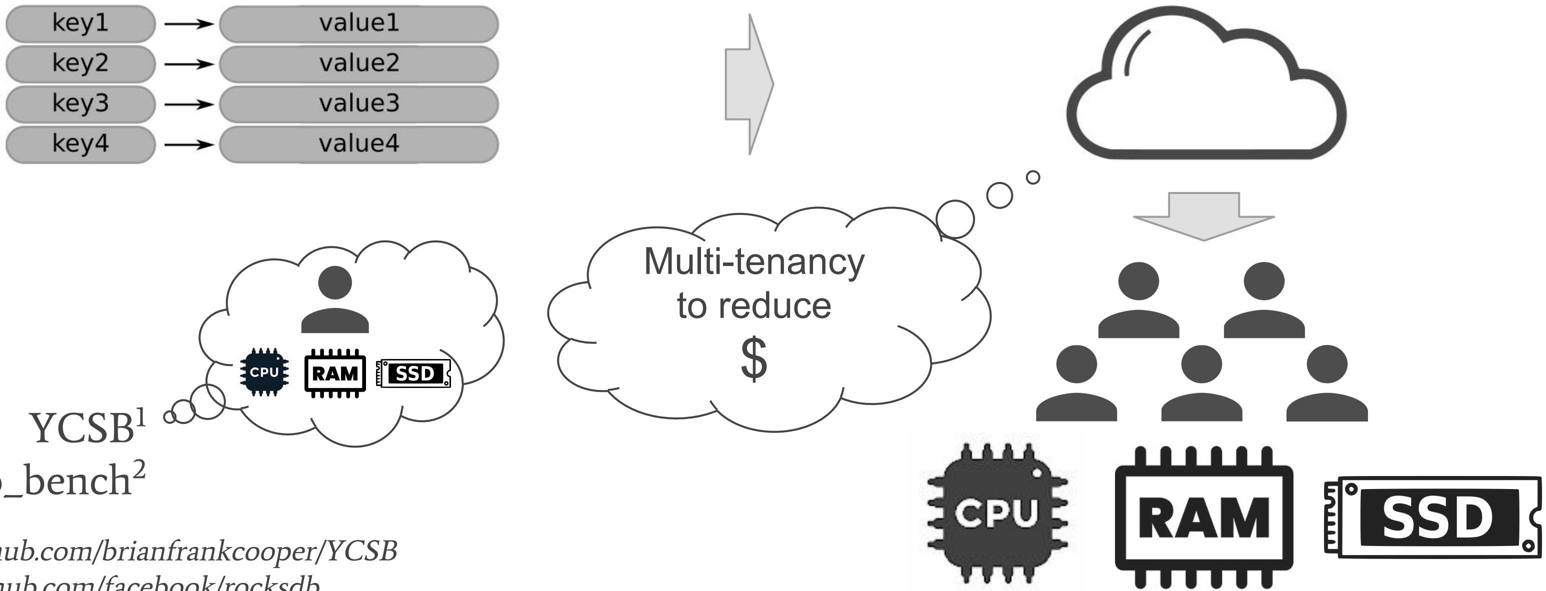
Cloud Computing



Key-Value Stores and Cloud Computing

Key-Value Stores

Cloud Computing

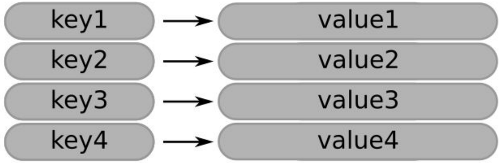


YCSB[1]
db_bench[2]

[1] *https://github.com/brianfrankcooper/YCSB*
[2] *https://github.com/facebook/rocksdb*

Our Focus

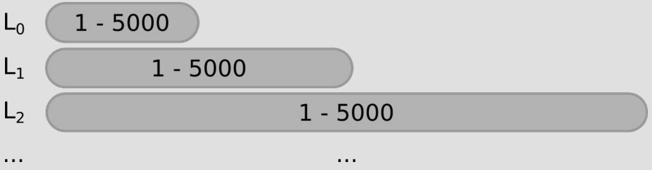
Key-Value Stores



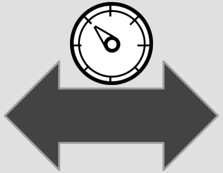
Cloud Computing

Implementation: RocksDB

LSM-tree



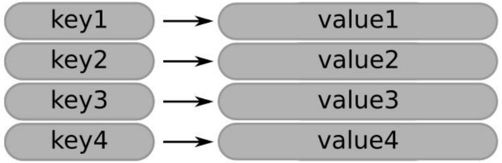
Pressure Scale



Target Resource: SSDs

Our Focus

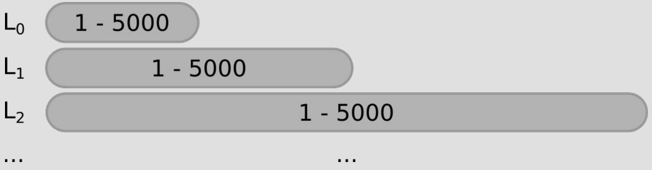
Key-Value Stores



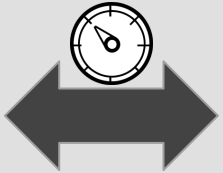
Cloud Computing

Implementation: RocksDB

LSM-tree



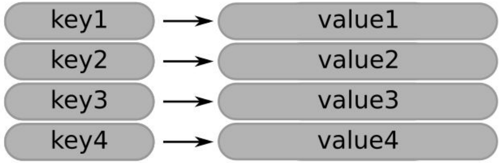
Pressure Scale



Target Resource: SSDs

Our Focus

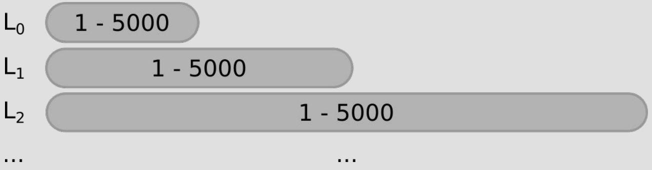
Key-Value Stores



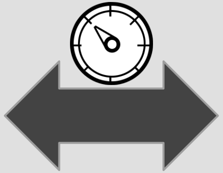
Cloud Computing

Implementation: RocksDB

LSM-tree



Pressure Scale



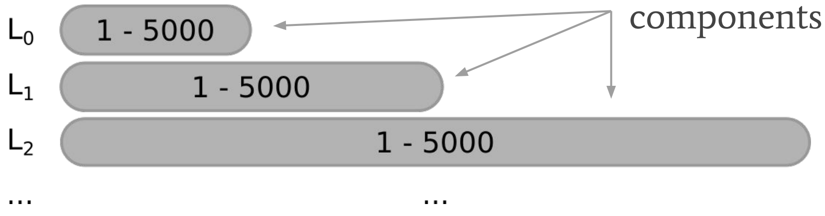
Target Resource: SSDs

Log-Structured Merge-Trees (LSM-trees)

Are composed of multiple levels, each one with one or more components (runs).

Merge policy: ***leveling***

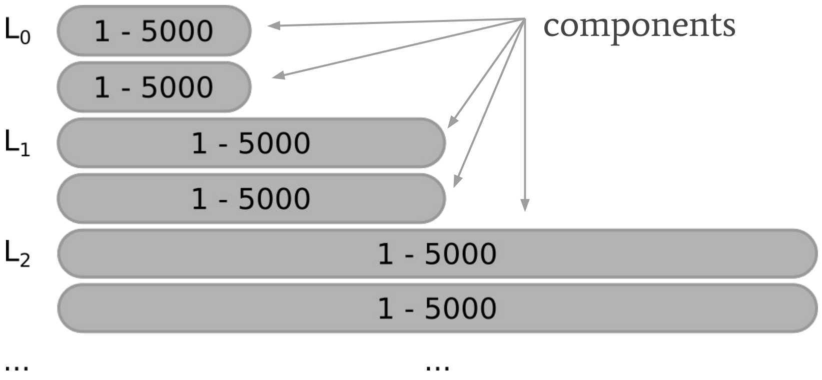
- One component per level (**L_0** a **L_L**).



$$T = \frac{\text{size of } L_{i+1}}{\text{size of } L_i}$$

Merge policy: ***tiering***

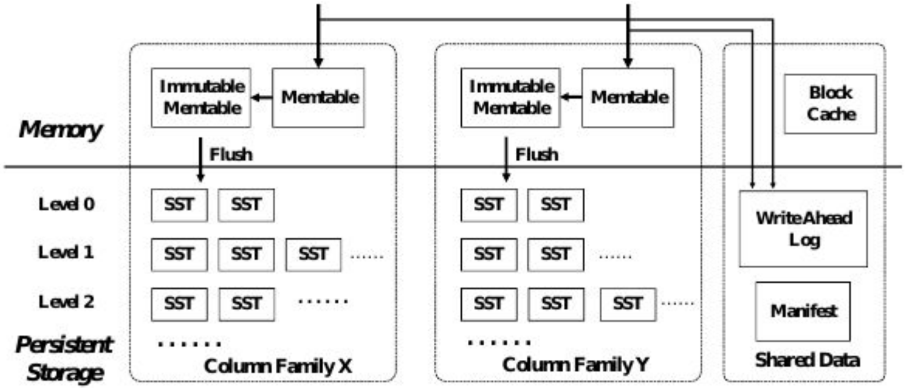
- Multiple components per level (**L_0** a **L_L**).
- Up to ***T*** components.



RocksDB

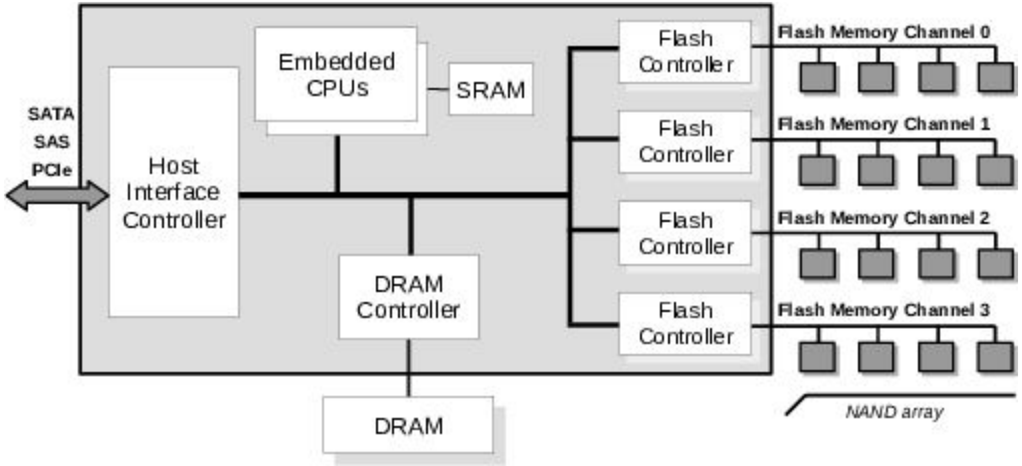
Basic Architecture

- Supports one or more LSM-trees (*Column Families*)
- Tiering merge policy: $\mathbf{L_0}$
- Leveling merge policy: $\mathbf{L_1}$, $\mathbf{L_2}$, ...



Zhichao Cao, Siying Dong, Sagar Vemuri, and David H C Du. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In FAST 2020.

Flash Devices



S. Kim, H. Oh, C. Park, S. Cho, and S. Lee, "Fast, energy efficient scan inside flash memory," in ADMS@VLDB 2011.

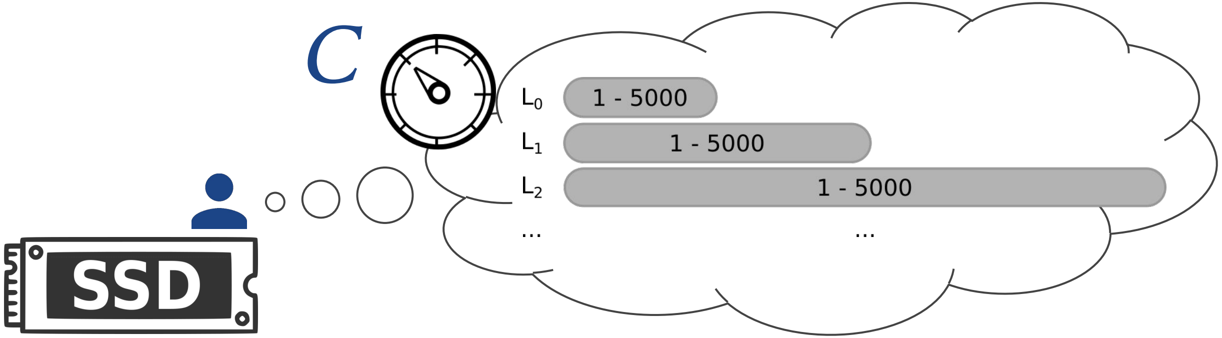
Flash Devices: Characteristics

- read and write operations at page level
- a page must be erased before written
- erase operations at block level (64 ~ 512 pages)
- writes must be sequential within a block
- limited number of erases per block
  (e.g., 5×10^4 up to 10^6 erases/block)

M. Bjørling, P. Bonnet, L. Bouganim, and N. Dayan, "The Necessary Death of the Block Device Interface," in CIDR, 2013.

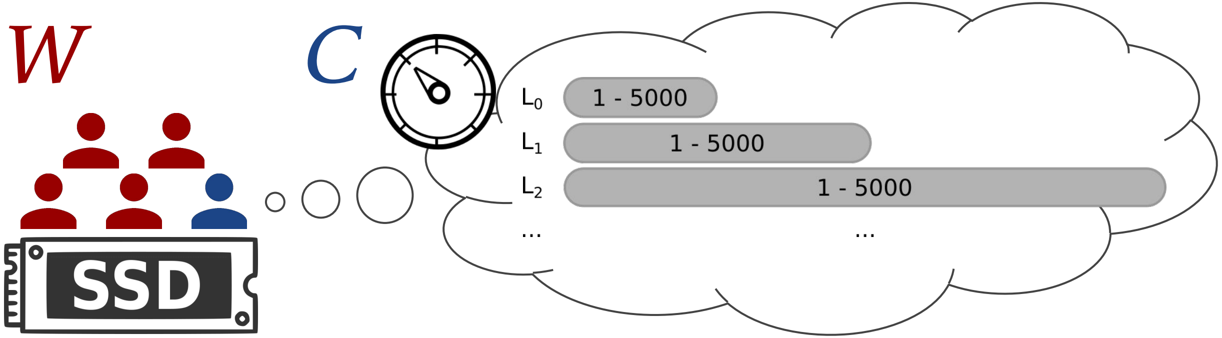
Pressure Scale

- C - target performance



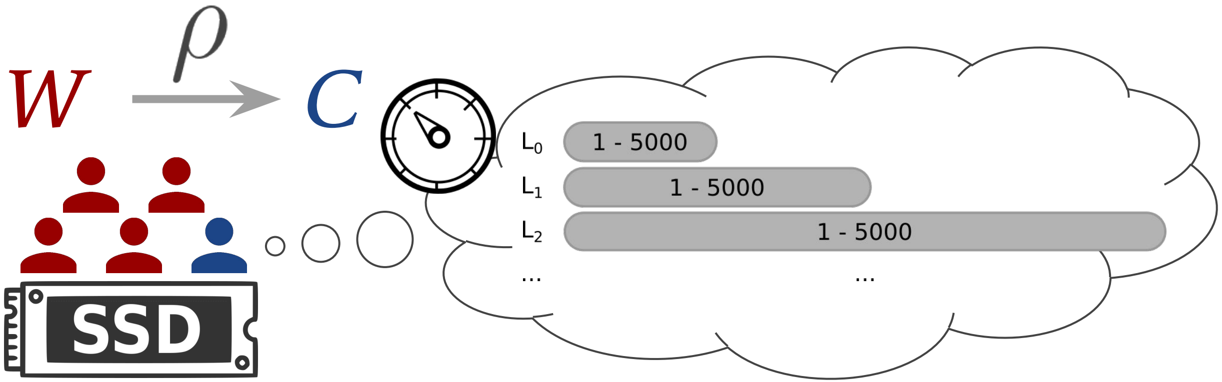
Pressure Scale

- C - target performance
- W - set of concurrent workloads



Pressure Scale

- C - target performance
- W - set of concurrent workloads
- $\rho : W \to C$ - pressure function



Pressure Scale

- $(C, \preceq_C)$ - linear order: $c_1 \preceq_C c_2$
  "c_1 is an inferior performance value than or equal to c_2"
- $(W, \succeq_W)$ - linear preorder: $w_1 \succeq_W w_2 \Leftrightarrow \rho(w_1) \preceq_C \rho(w_2)$
  "w_1 exerts more or the same pressure than w_2"
- w_0 - no concurrent workloads

Pressure Scale

- C as the average throughput of the key value-store

- $\preceq_C$ as the numeric relation $\leq$
  Once lower throughput is equivalent to inferior performance value:

$$w_2 \succeq_W w_1 \Leftrightarrow \rho(w_2) \leq \rho(w_1)$$

- Produce $w_1, w_2, \ldots, w_n \in W$ so that $w_0 \preceq_W w_1 \preceq_W w_2 \cdots \preceq_W w_n$

W: Access_time3 Instances

A configurable microbenchmark:

- file_size: 10 GiB (on the same storage device of the key-value store)
- write_ratio (wr): ratio between write and read operations
    - 0 : 100% reads
    - 1 : 100% writes
- random_ratio (rr): ratio between random and sequential access patterns
    - 0 : 100% sequential
    - 1 : 100% random
- block_size (bs): size of each access operation (KiB)
- Flags O_DIRECT+O_DSYNC

Access_time3 Instances and $W^{bs=X}$

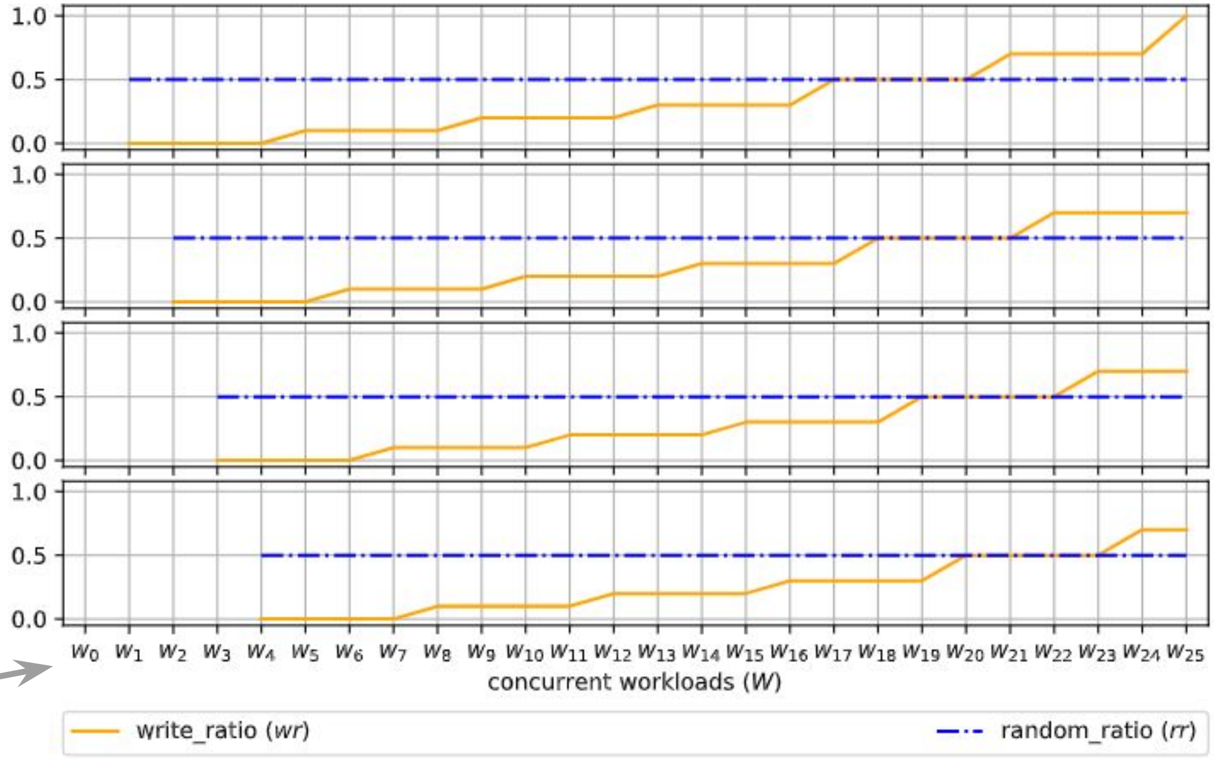
block_size

access_time3 instance 1:

access_time3 instance 2:

access_time3 instance 3:

access_time3 instance 4:



$W^{bs=X}$ instances

Access_time3 Instances and $W^{bs=X}$

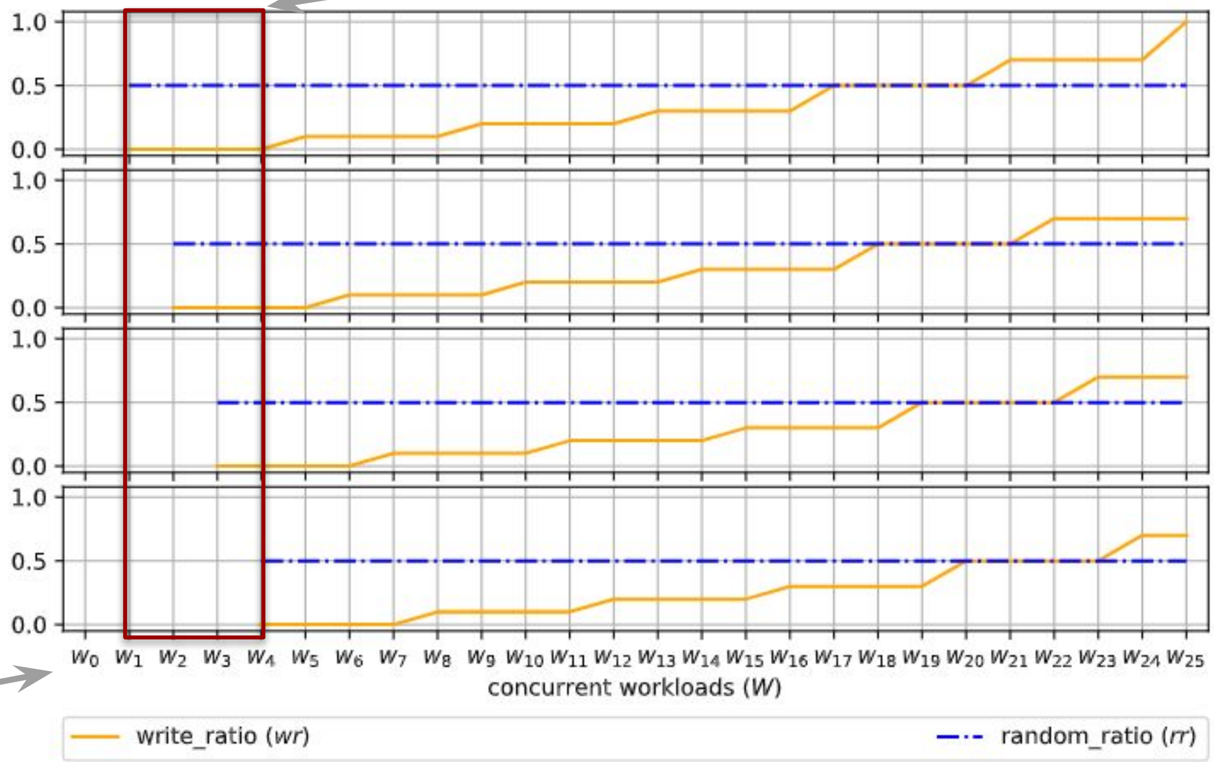
READ workloads

access_time3 instance 1:

access_time3 instance 2:

access_time3 instance 3:

access_time3 instance 4:



$W^{bs=X}$ instances

Access_time3 Instances and $W^{bs=X}$

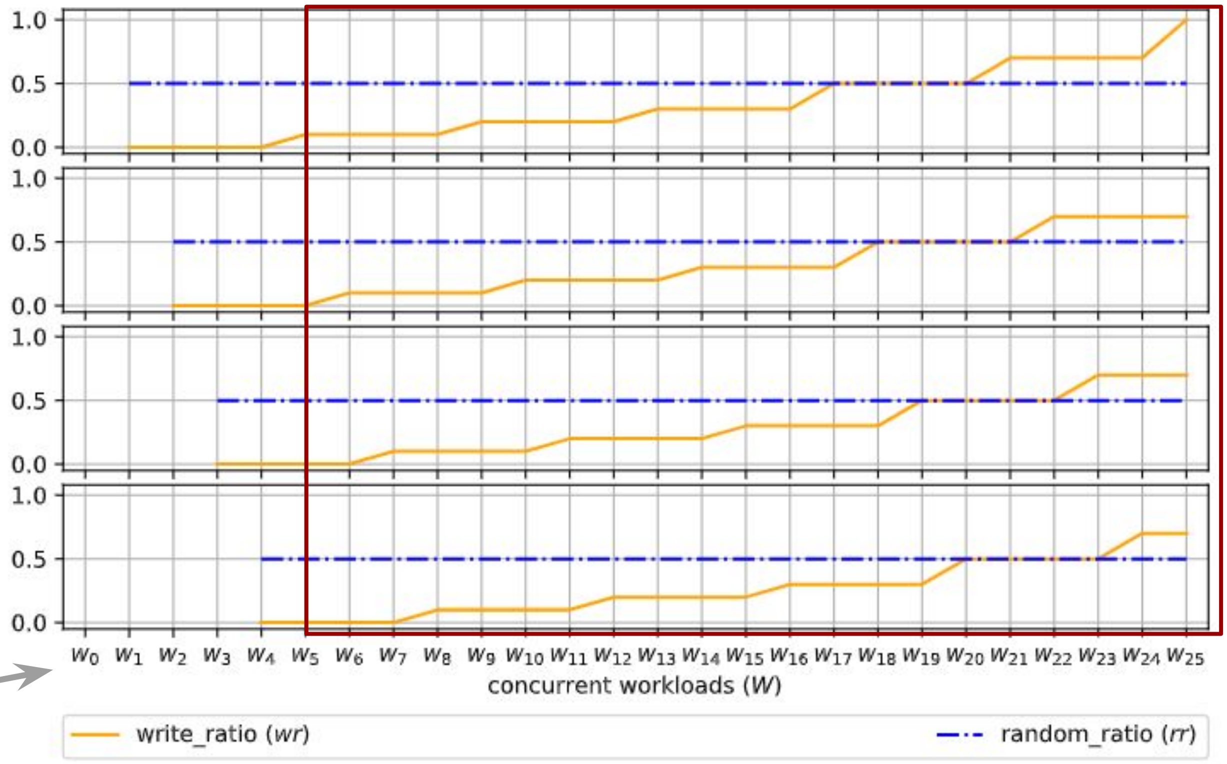
READ/WRITE workloads

access_time3 instance 1:

access_time3 instance 2:

access_time3 instance 3:

access_time3 instance 4:

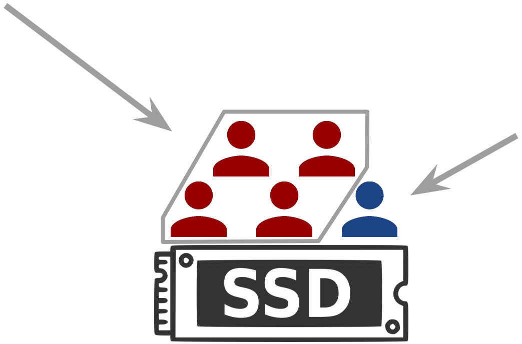


$W^{bs=X}$ instances

Experiments

$W^{bs=x}$ (in KiB)

- $x \in \{4, 8, 16, 32, 64, 128, 256, 512\}$

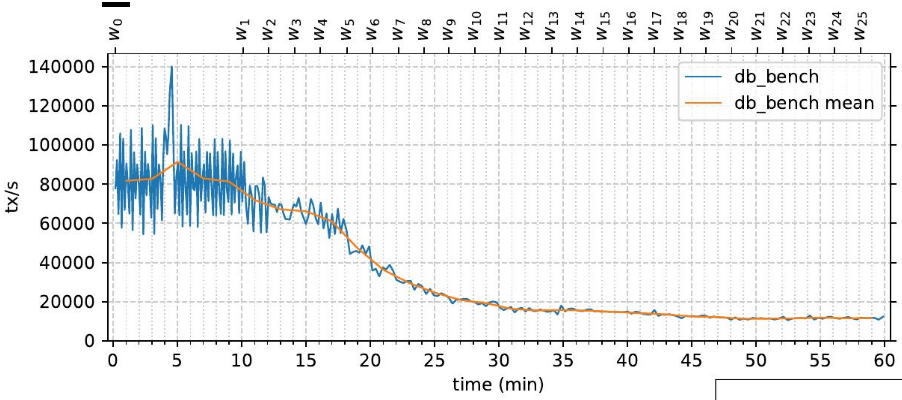


Samsung 970 EVO NVMe 512GB

C : *average throughput*

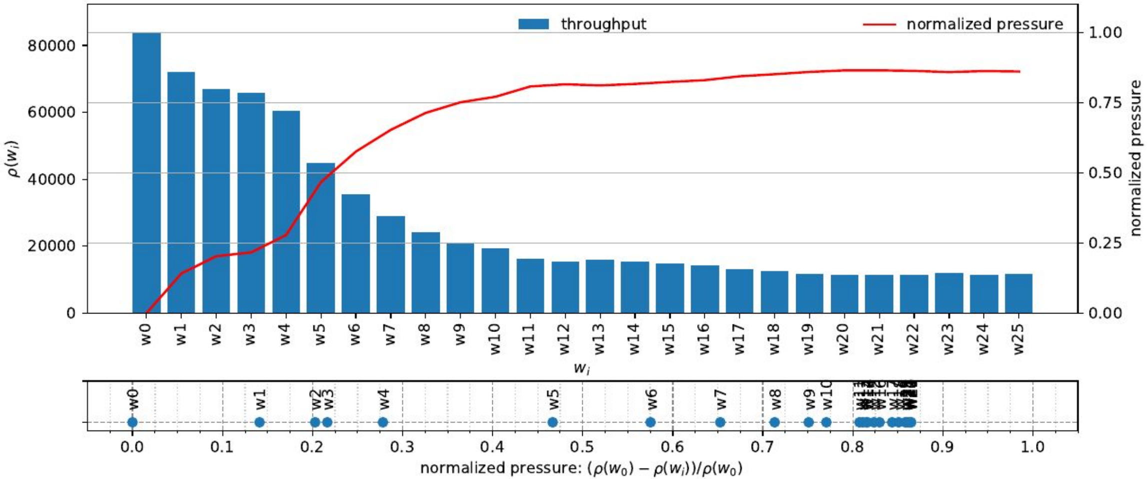
- YCSB workload:
  - A (50% get, 50% put)
  - B (95% get, 5% put)
- db_bench workload readwhilewriting

db_bench and $W^{bs=512KiB}$

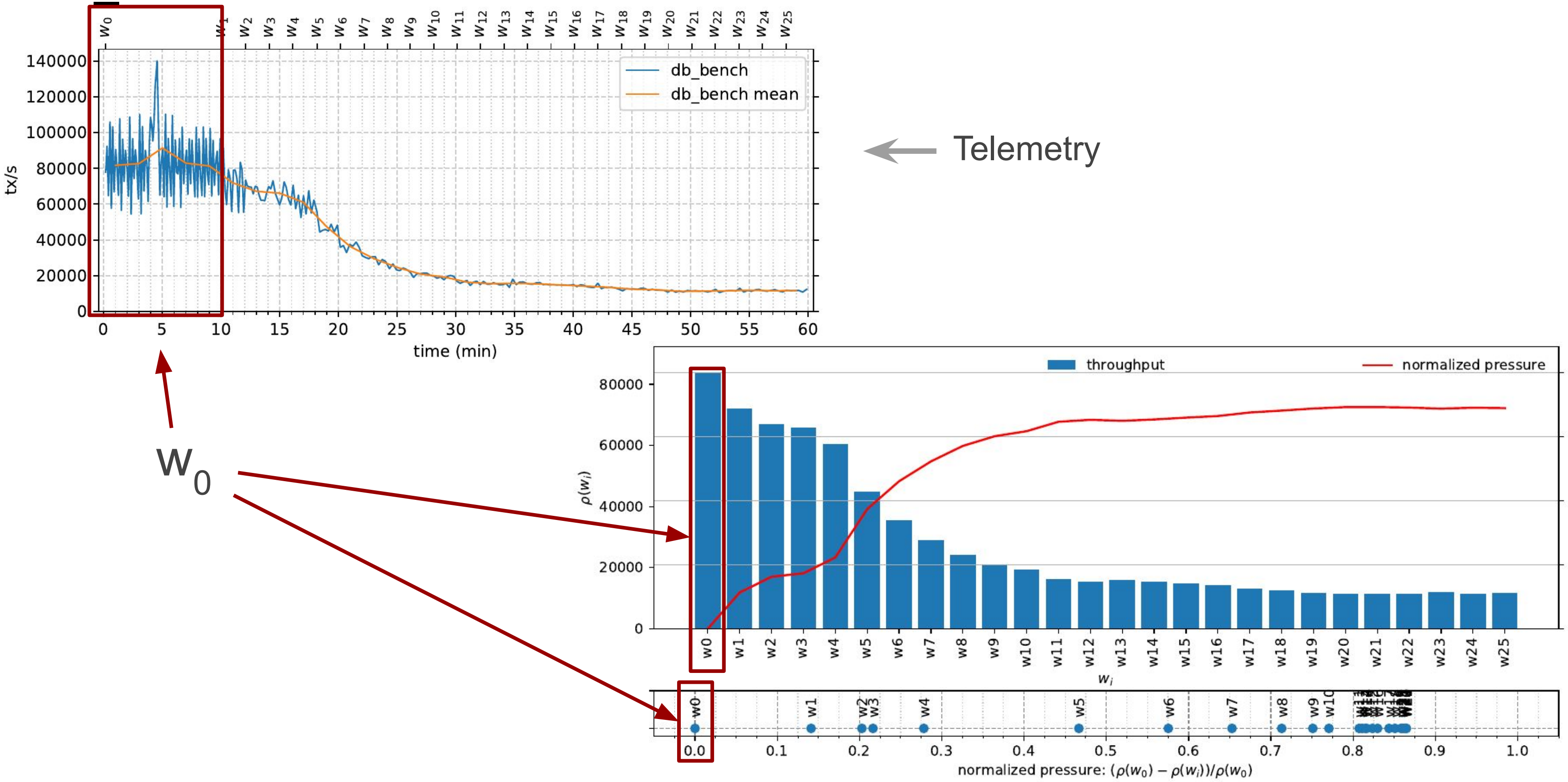


Telemetry

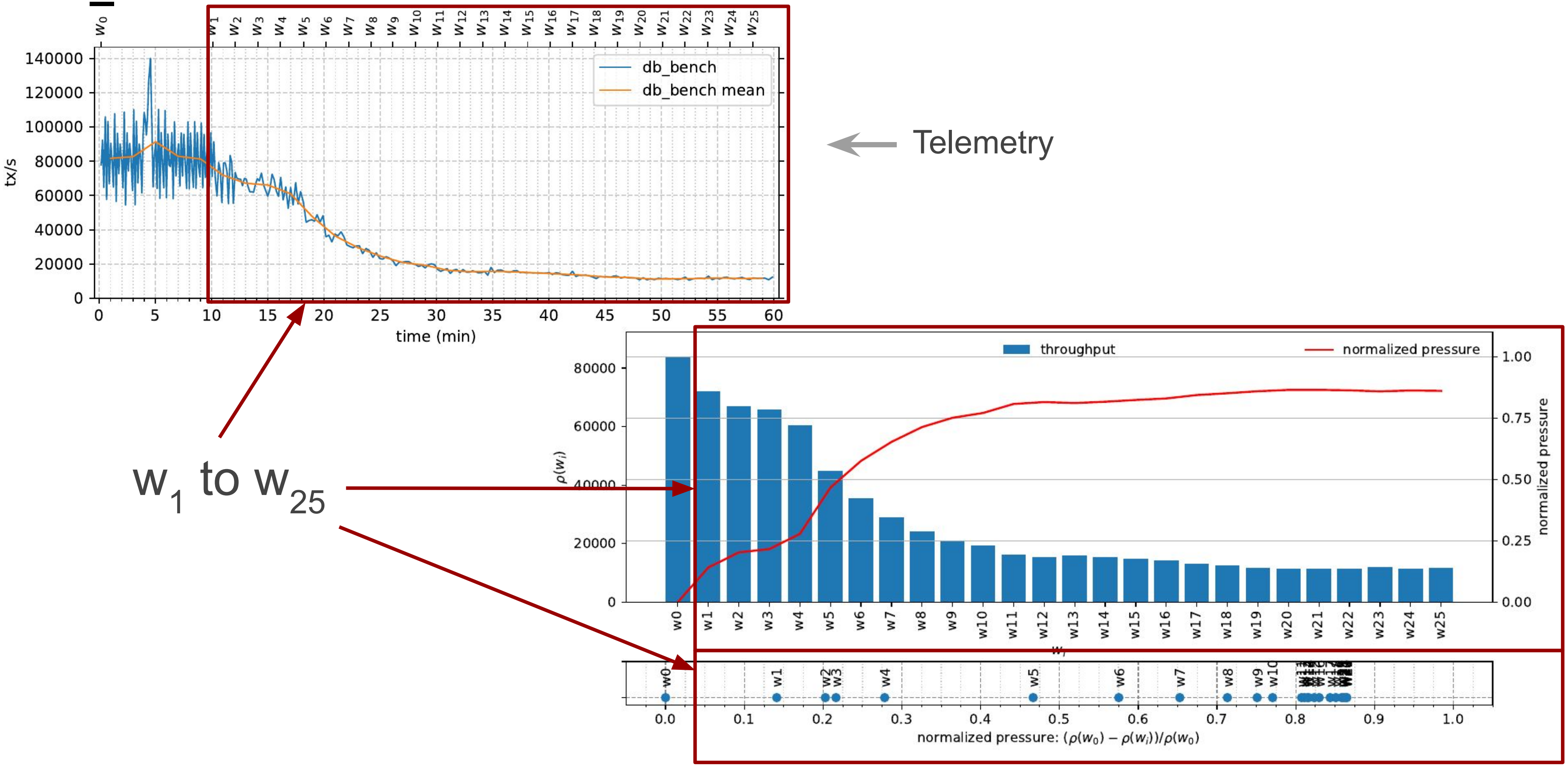
Pressures



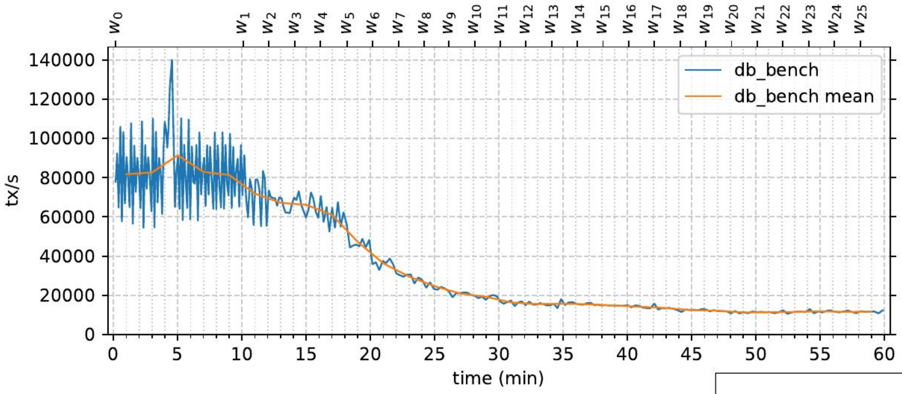
db_bench and $W^{bs=512KiB}$



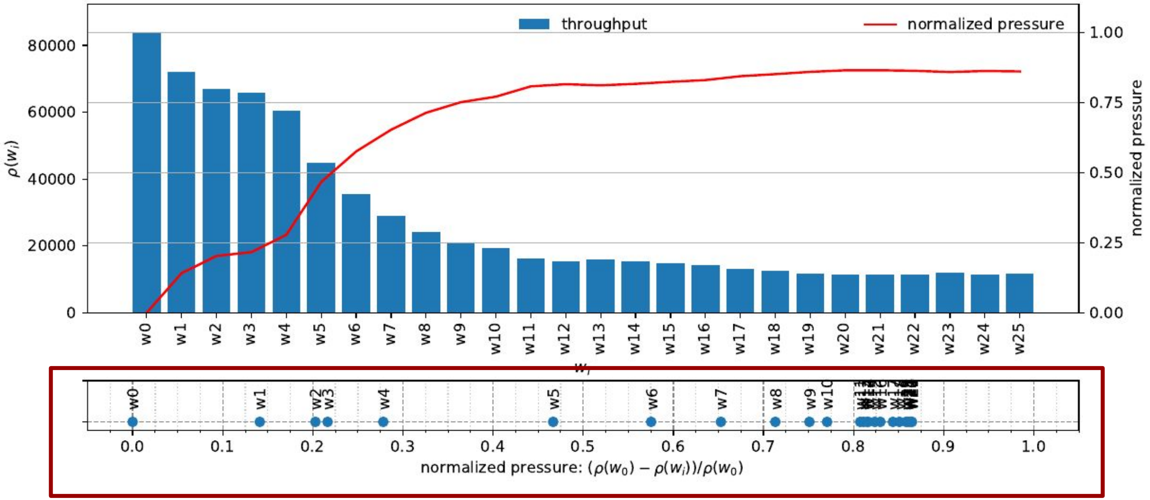
db_bench and $W^{bs=512KiB}$



db_bench and $W^{bs=512KiB}$



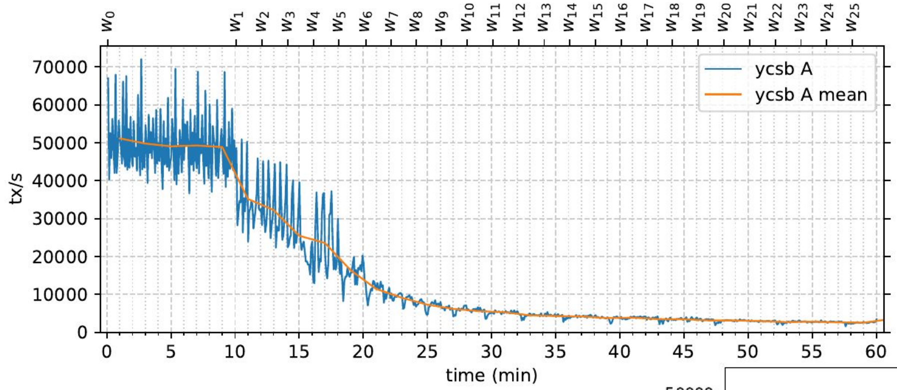
Telemetry



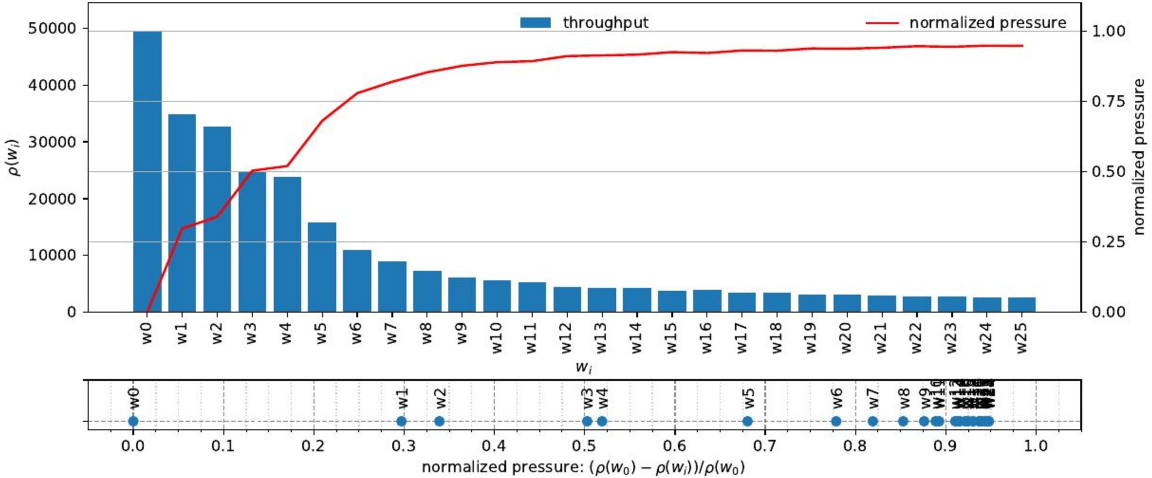
Normalized Pressures

$$\frac{\rho(w_0) - \rho(w_i)}{\rho(w_0)}$$

YCSB workload A and $W^{bs=512KiB}$



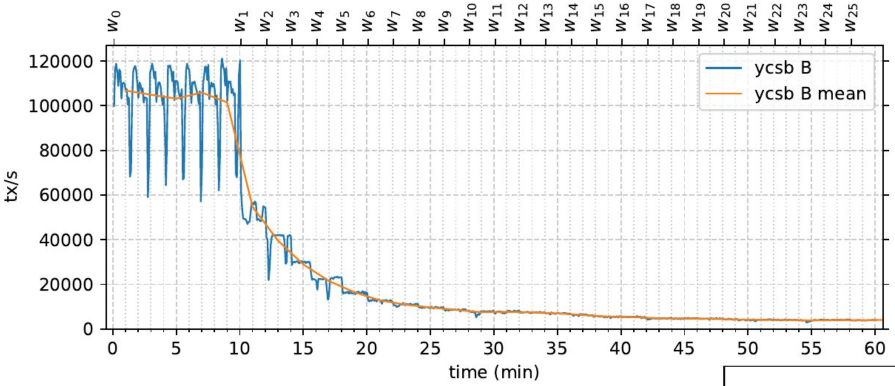
Telemetry



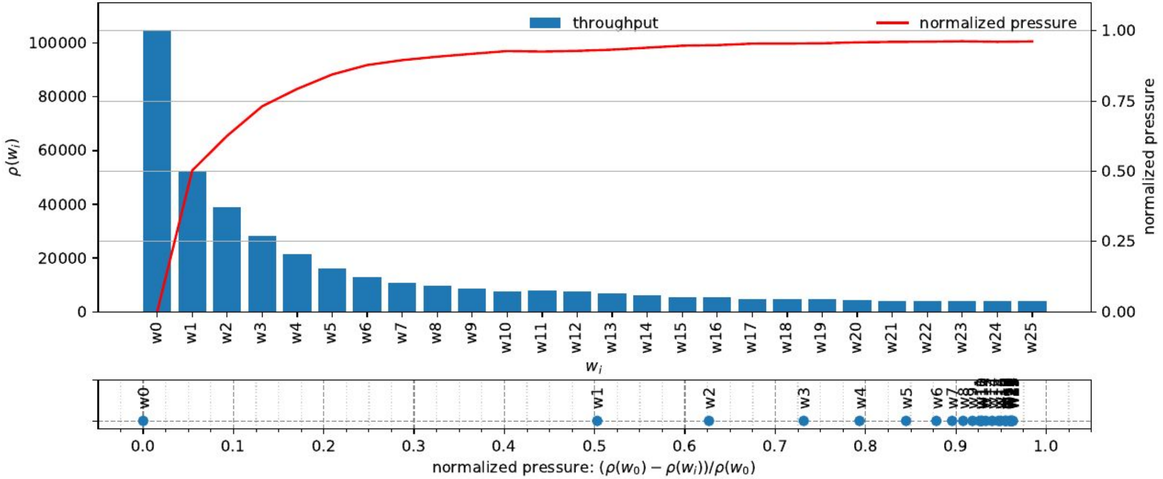
Normalized Pressures

$$\frac{\rho(w_0) - \rho(w_i)}{\rho(w_0)}$$

YCSB workload B and $W^{bs=512KiB}$



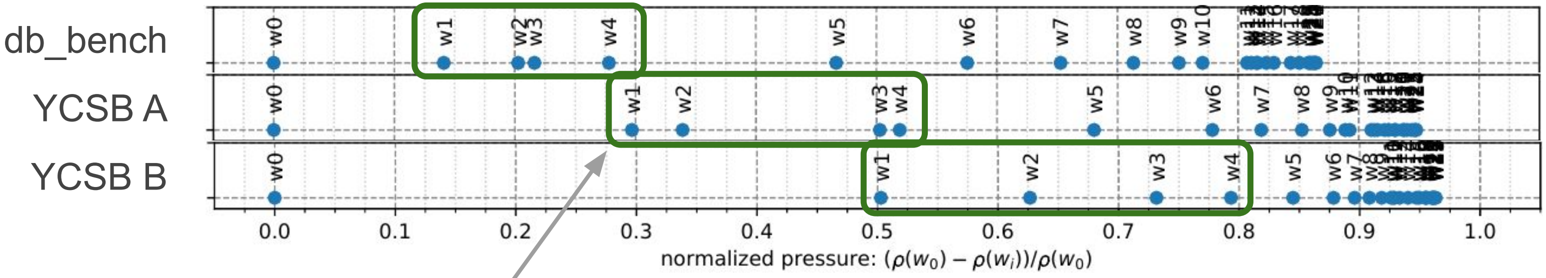
Telemetry



Normalized Pressures

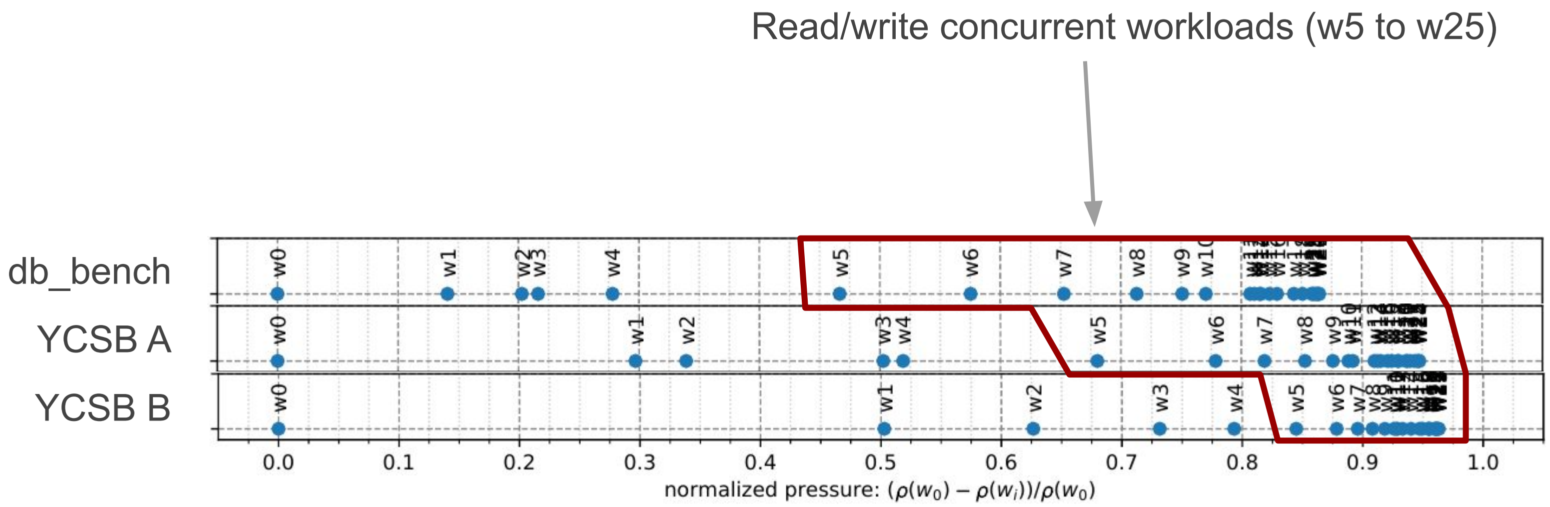
$$\frac{\rho(w_0) - \rho(w_i)}{\rho(w_0)}$$

Pressure scale: $W^{bs=512KiB}$

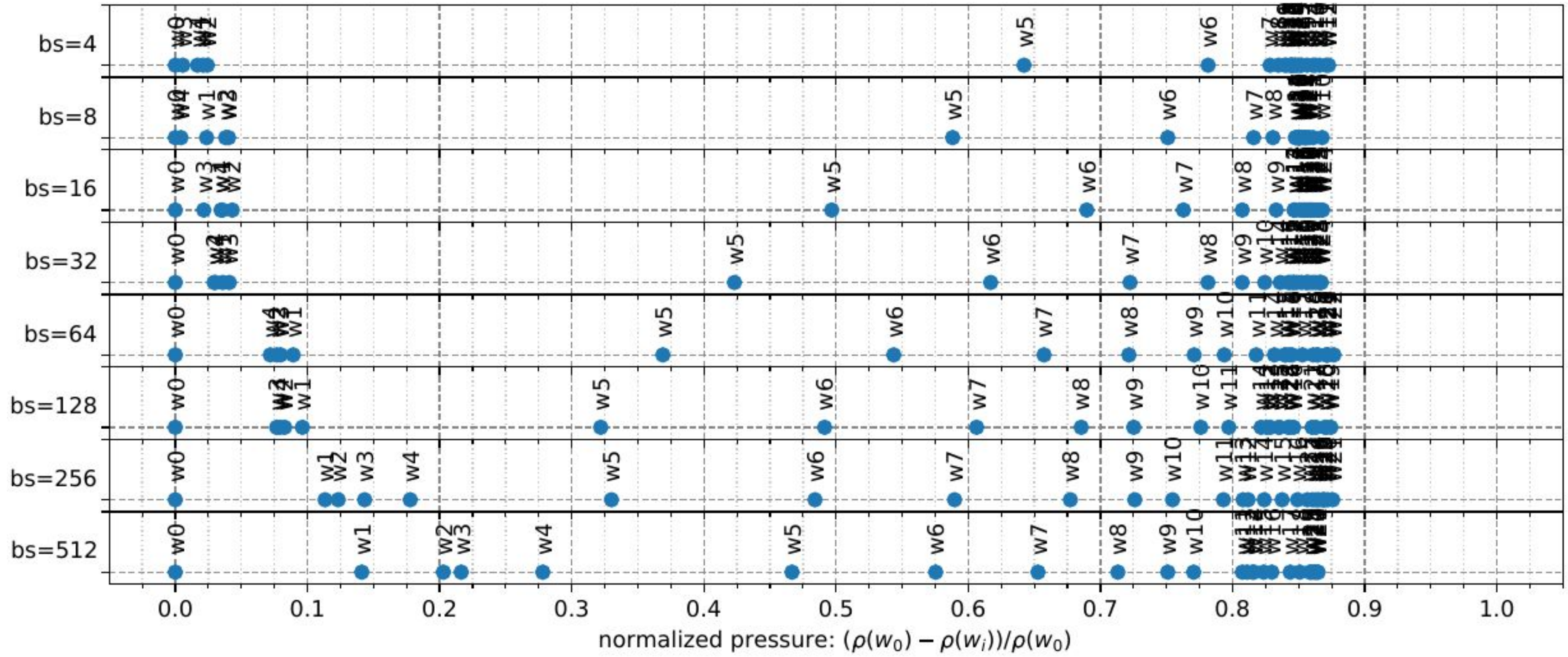


Read-only concurrent workloads (w1 to w4)

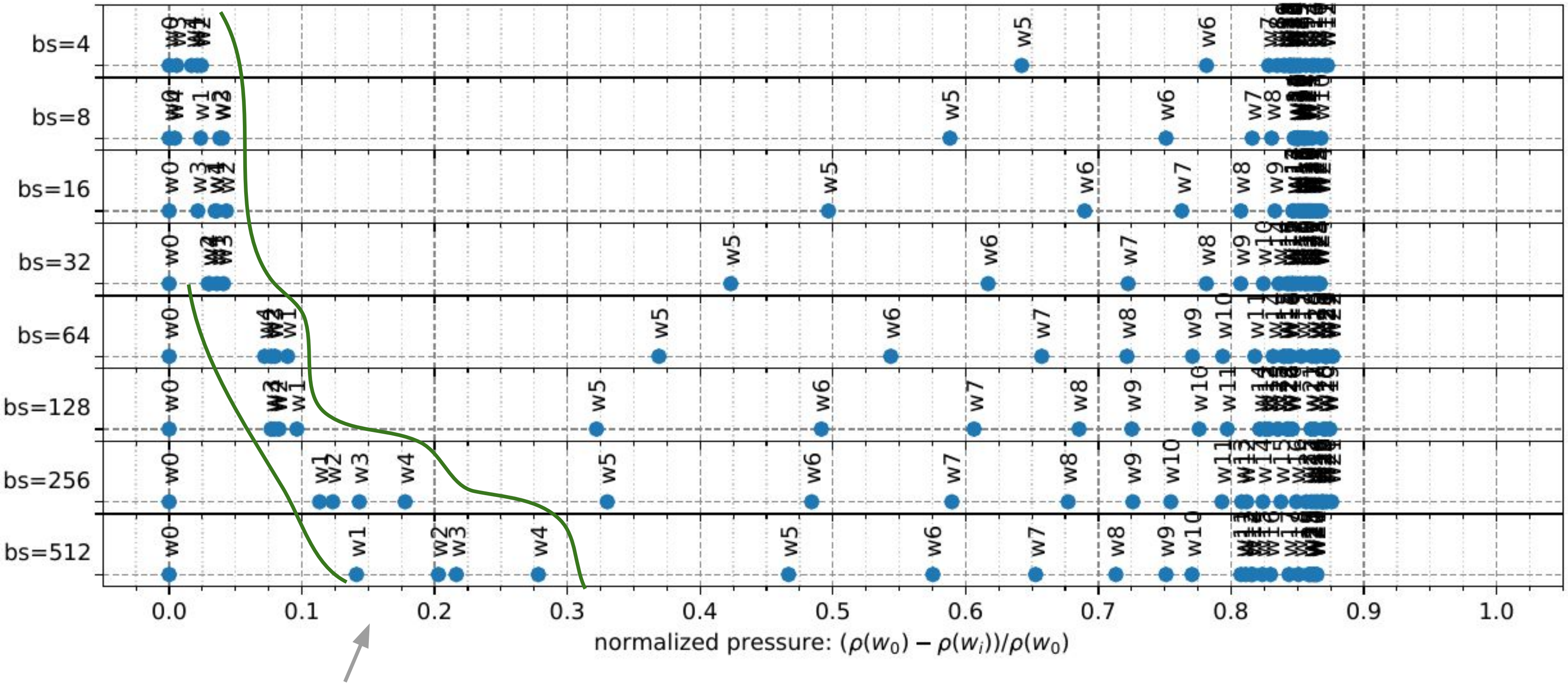
Pressure scale: $W^{bs=512KiB}$



Pressure scale: db_bench



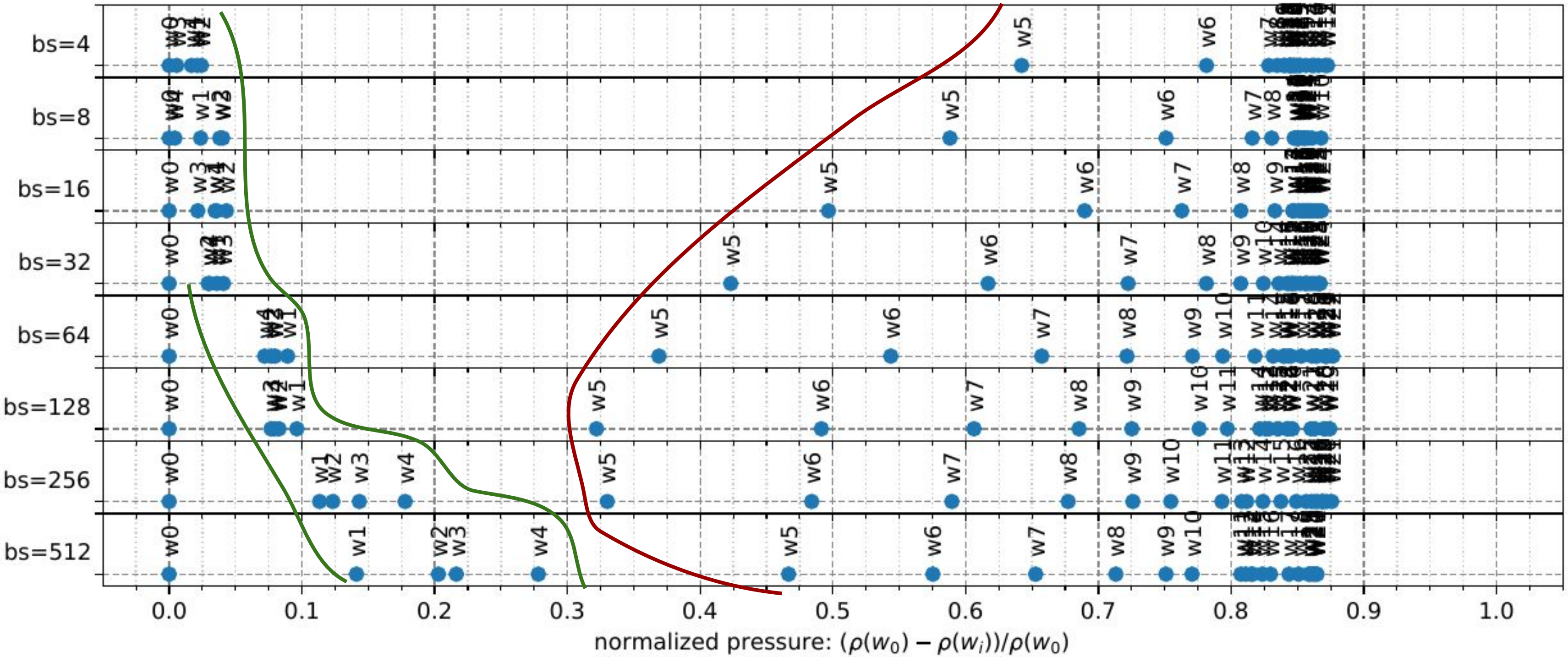
Pressure scale: db_bench



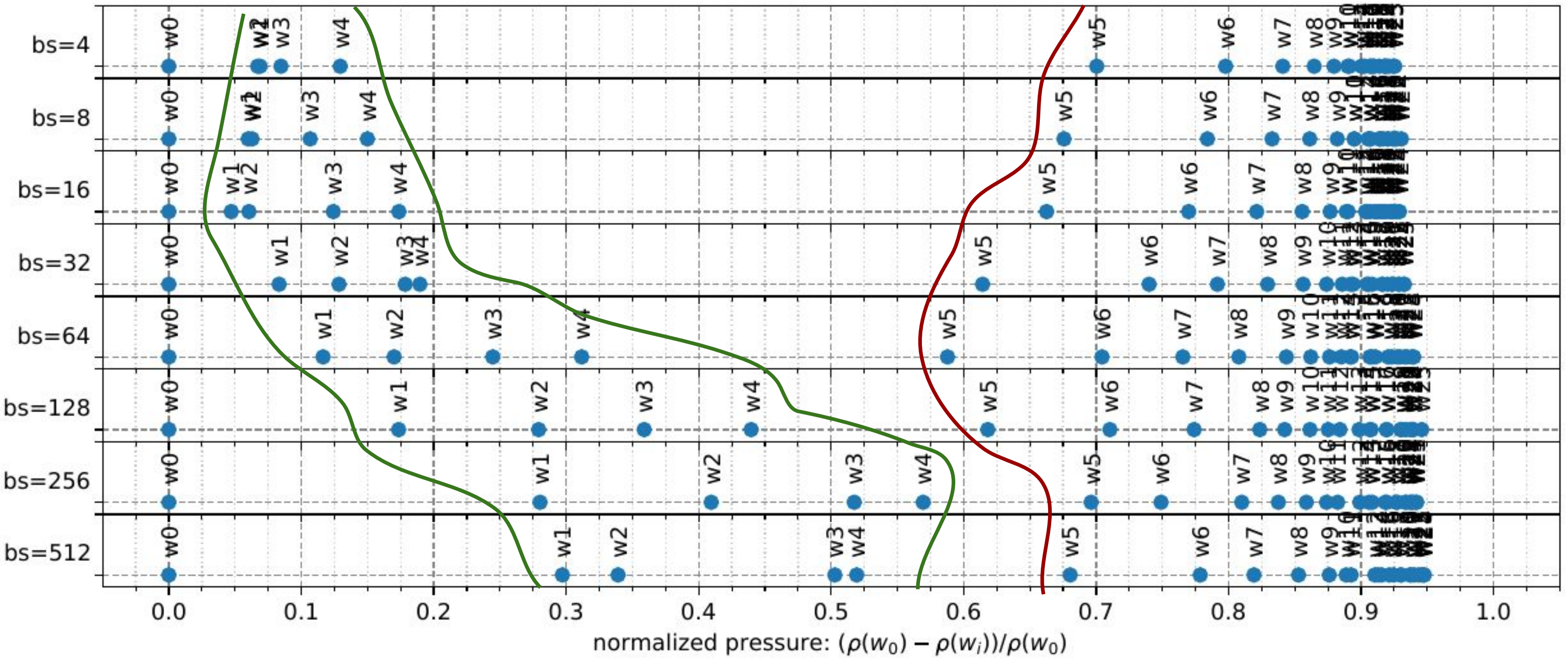
Read-only concurrent workloads (w1 to w4)

Pressure scale: db_bench

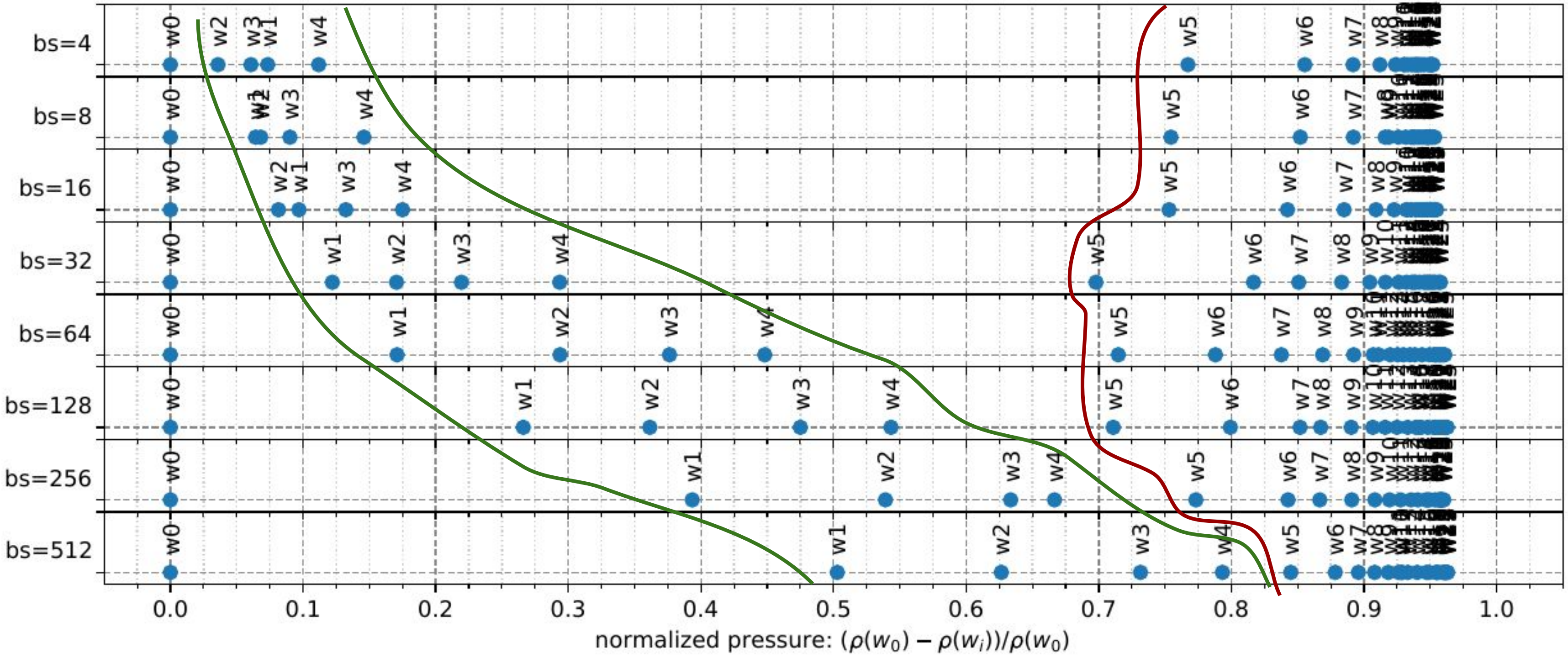
Read/write concurrent workloads (w5 to w25)



Pressure scale: YCSB workload A



Pressure scale: YCSB workload B



Summary

- The pressure produced by our concurrent workloads varied according to the workload submitted to the key-value store.

Summary

- Better co-location options:
  - Concurrent **read-only** workloads with small read requests (<= 16 KiB)
    - *Internal parallelism of the storage device*
  - Concurrent **write** workloads with intermediate write requests (>= 32 and <= 128 or <=258 KiB, depending on the key-value workload)
- Concurrent write workloads may represent **serious performance issues** for the key-value store.
  - *Small concurrent write requests: Possible contentions related to either some synchronization mechanism or the storage device's garbage collector.*

Project Status and Future Work

- Implementing:
  - Retrieve more details about the hardware and OS states.
    - Internal state of the storage device (using smart and nvme)
  - Retrieve more details about the internal state of the key-value store (LSM-tree)
    - How the key-value store could minimize the performance impact of concurrent workloads?

- Potential further evaluations:
  - Compare different SSDs
  - Random *versus* sequential concurrent workloads
  - Compare different I/O schedulers

Project's Repository and Contact

https://github.com/alange0001/rocksdb_test

alange@inf.ufpr.br