KVAccel: A Novel Write Accelerator for LSM-Tree-Based KV Stores with Host-SSD Collaboration

Kihwan Kim[1], Hyunsun Chung[1], Seonghoon Ahn[1], Junhyeok Park[1], Safdar Jamil[1], Hongsu Byun[1], Myungcheol Lee[2], Jinchun Choi[2], Youngjae Kim[1]

The 39th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Milan, Italy, June 3-7, 2025

Contents

Background

LSM-tree based Key-Value Stores

DISCOS

- Log-Structured Merge-Tree(LSM-tree)
  - Designed for write-intensive workloads
  - Optimized for large-scale data
  - Out-of-place updates
  - Sequential batch operations

RocksDB [1]

levelDB [2]

ZippyDB [3]

[1]: Facebook, "RocksDB" https://rocksdb.org, 2012
[2]: Google, "LevelDB" https://github.com/google/leveldb, 2017
[3]: Meta, "ZippyDB" https://engineering.fb.com/2021/08/06/core-infra/zippydb/, 2021

LSM-tree based Key-Value Stores

- LSM KVS(e.g. RocksDB) stores data in an append-only manner in the active MemTable
- Data in MemTable is moved to and managed on disk through background jobs(Flush, Compaction)

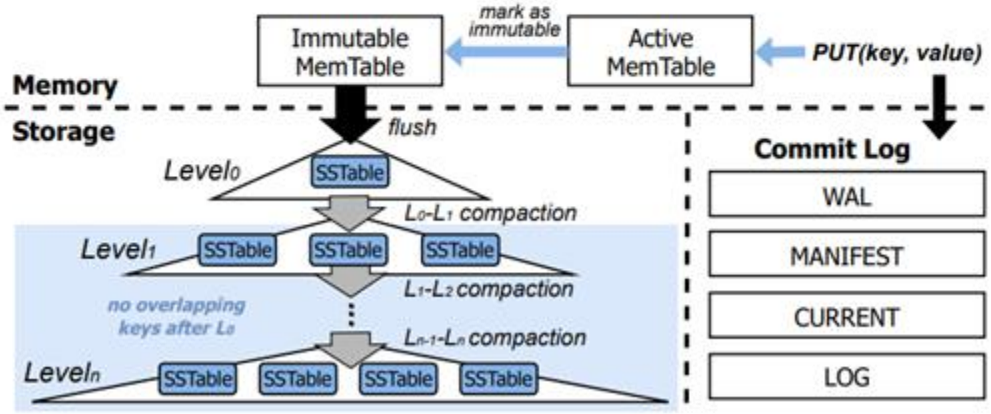


Fig. 1: An architecture of LSM-tree.

Write Stall Problem

- Write Stall: write operation blocked, due to bottlenecks in Flush, Compaction
- In RocksDB, Write stall occurs under these 3 scenarios[4][5]
    - Incoming Writes > Flush
    - Flush > Level 0 to Level 1 Compaction
    - Pending deep level compaction size becomes heavier

[4]: SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores, Oana Balmau et al., USENIX ATC'19
[5]: ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance, Jinghuan Yu et al. (USENIX FAST'23)

Existing Work: ADOC[5]

- In three types of overflow scenarios, ADOC alleviates write stalls by adjusting two tuning knobs
- Two tuning knobs: # of Compaction threads, MemTable size

	# of Compaction Threads	**MemTable Size**
Incoming Writes > Flush	↓	↑
Flush > Level 0 to Level 1 Compaction	↑	
Pending deep level compaction size becomes heavier	↑	↓

[5]: ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance, Jinghuan Yu et al. (USENIX FAST'23)

Existing Work: ADOC[5]

- In three types of overflow scenarios, ADOC alleviates write stalls by adjusting two tuning knobs
- Two tuning knobs: # of Compaction threads, MemTable size

1. Not an immediate remedy → Write stalls still occur

2. Requires *Slowdown methods while accelerating compaction*

Pending deep level compaction size becomes heavier	↑	↓

[5]: ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance, Jinghuan Yu et al. (USENIX FAST'23)

Motivation

Observation 1.
Slowdowns[6]: The Inefficient Write Stall Solution

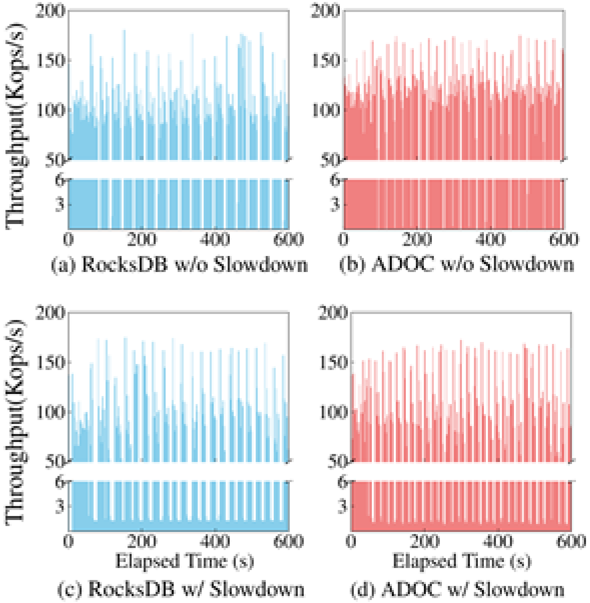
- RocksDB uses the *slowdown*[6] method to prevent user writes from becoming completely blocked.
- The state of the art solution ADOC[5] also uses *slowdowns*.

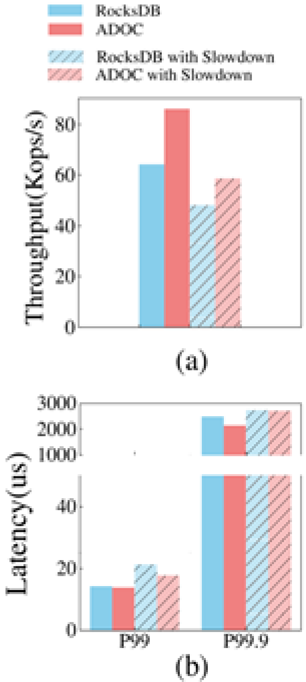
➡ Both RocksDB and ADOC[5] ultimately fall back to using *slowdown* to avoid a write stall.

[5]: ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance, Jinghuan Yu et al. (USENIX FAST'23)
[6]: https://github.com/facebook/rocksdb/wiki/Write-Stalls

Observation 1.
Slowdowns[6]: The Inefficient Write Stall Solution

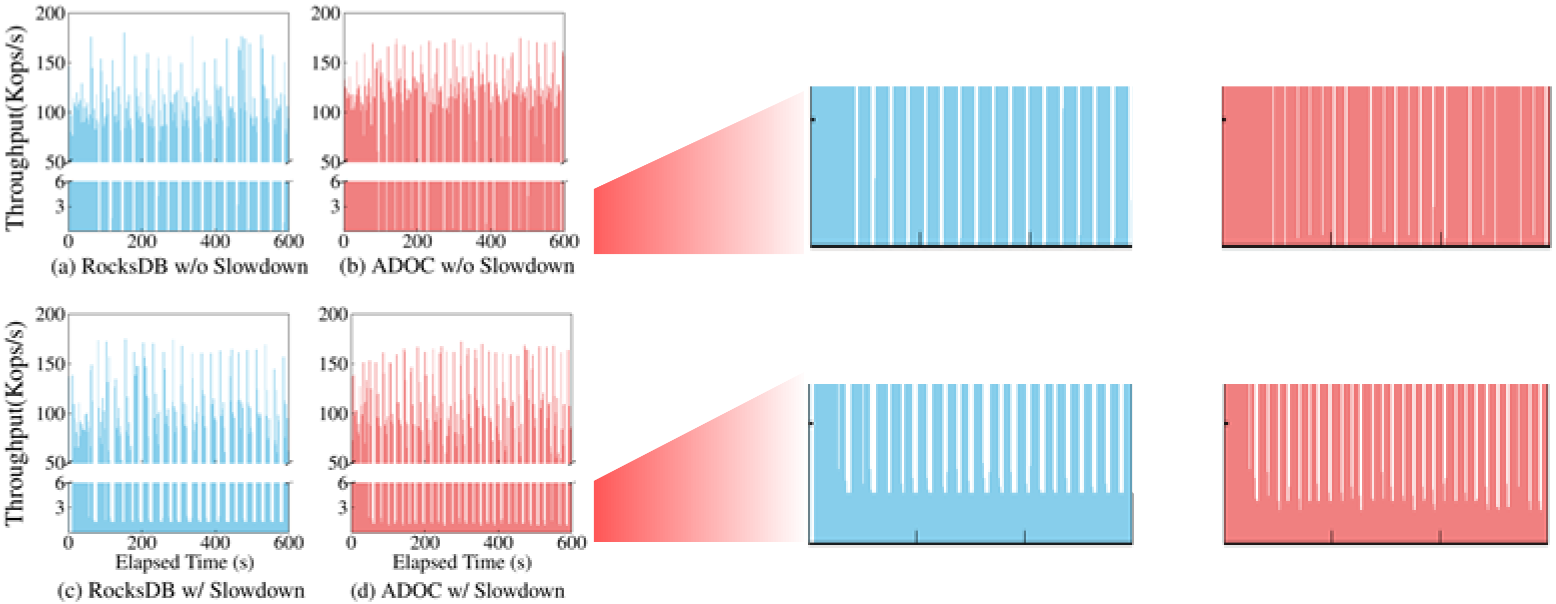
- *Slowdowns*, while preventing a complete write stall from occurring, harms overall performance.





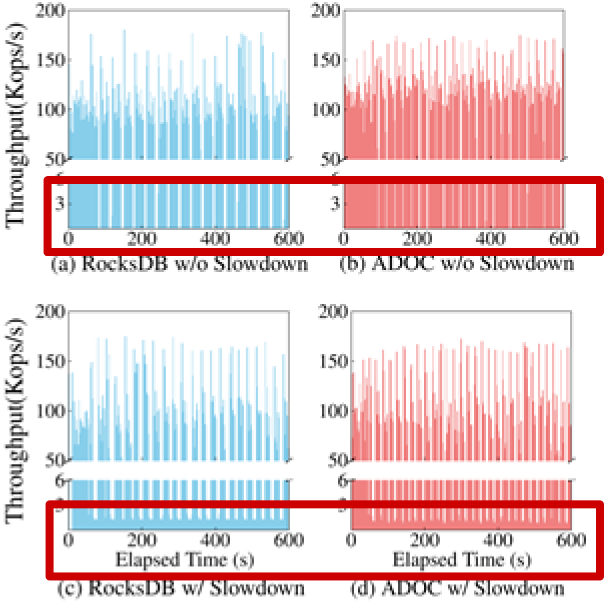
Observation 1.
Slowdowns[6]: The Inefficient Write Stall Solution

- *Slowdowns*, while preventing a complete write stall from occurring, harms overall performance.

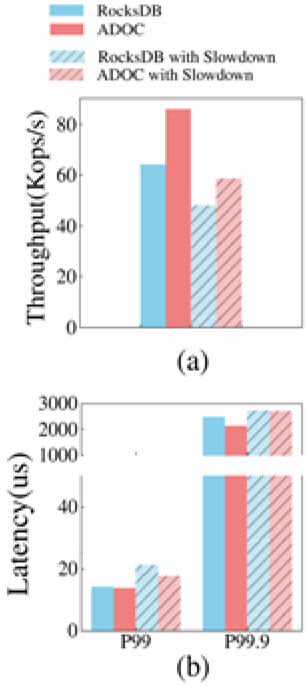


Observation 1.
Slowdowns[6]: The Inefficient Write Stall Solution

- *Slowdowns*, while preventing a complete write stall from occurring, harms overall performance.



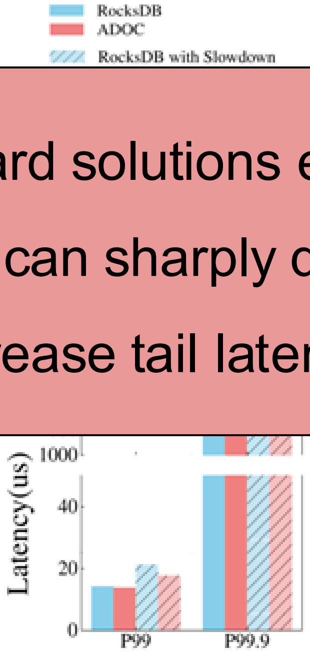
I/O service is uninterrupted thanks to slowdowns preventing write stalls...



...At the cost of overall throughput and latency.

Observation 1.
Slowdowns[6]: The Inefficient Write Stall Solution

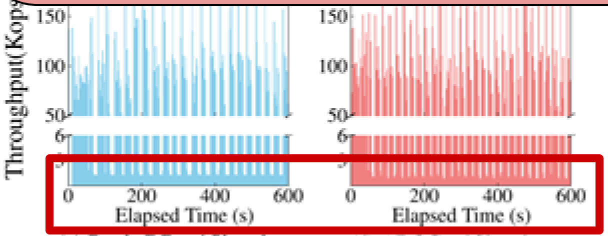
- *Slowdowns*, while preventing a complete write stall from occurring, harms overall performance.



(b)

Both state-of-the-art and industry-standard solutions employ write slowdowns to prevent write stalls, which can sharply degrade over throughput and significantly increase tail latency.

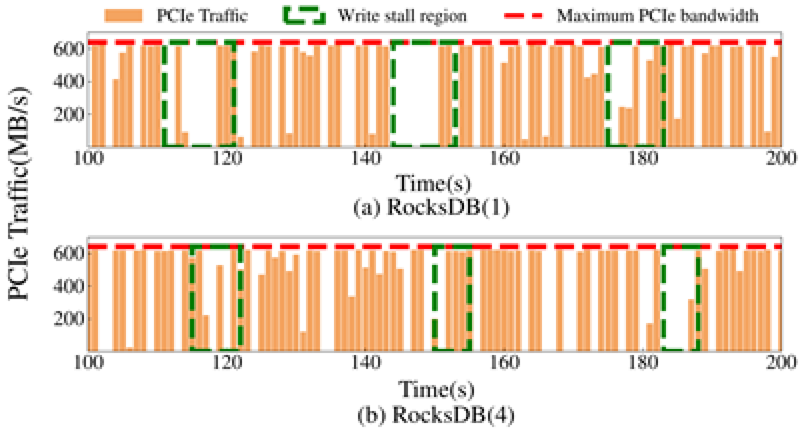
stalls...

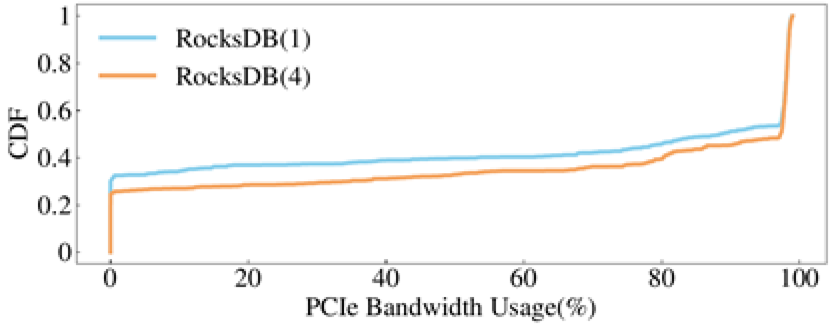


(c) RocksDB w/ Slowdown (d) ADOC w/ Slowdown

Observation 2.
Under-utilization of PCIe Bandwidth

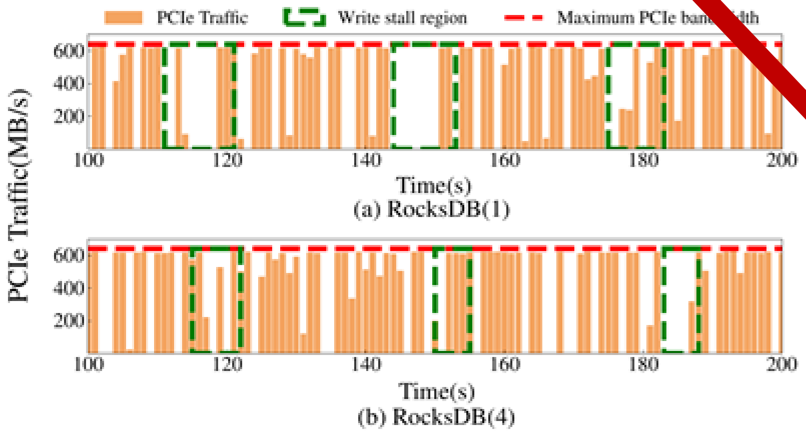
- PCIe Traffic drop sharply during a write stall, implying inefficient device resource usage.

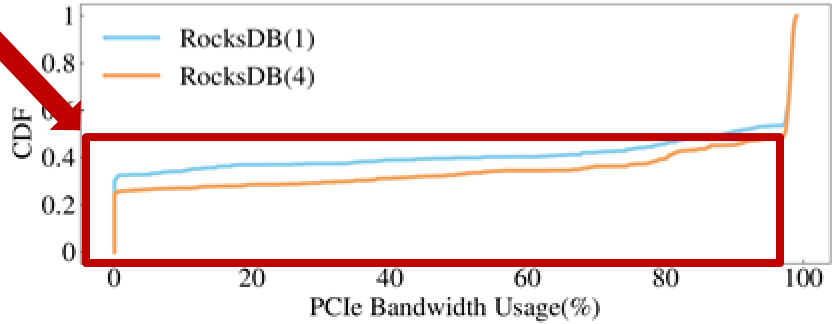




Observation 2.
Under-utilization of PCIe Bandwidth

- PCIe Traffic drop sharply during a write stall, implying inefficient device resource usage.
  - RocksDB is shown to leave up to **90%** of available PCIe bandwidth around **50%** of the time during a write stall.

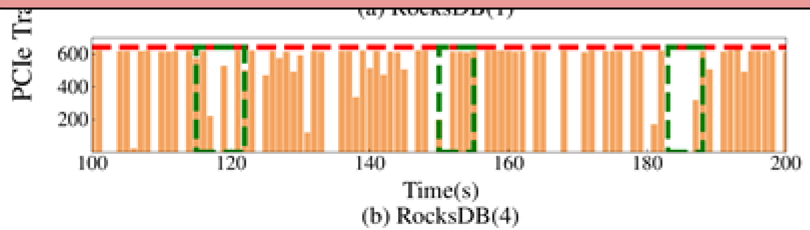


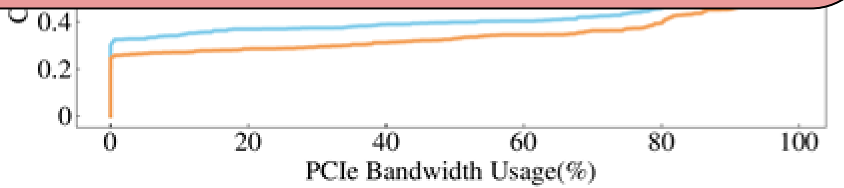


Observation 2.
Under-utilization of PCIe Bandwidth

- PCIe Traffic drop sharply during a write stall, implying inefficient device resource usage.

PCIe bandwidth is under-utilized during write stalls in industry standard LSM-KVS due to the compaction operation blocking device I/O.





The status quo

- **Observation 1.** ultimately leads to the following options for write stalls.

Slowdowns	VS	Allowing Write Stalls
• Maintains I/O service at all times • Overall throughput and latency penalty due to said slowdowns		• Overall throughput and latency conserved • Complete interrupts in I/O service as write stalls are allowed to occur.

- **Observation 2.** reveals an unexploited resource to help mitigate write stalls and increase performance without sacrificing system resources: underutilized PCIe and device bandwidth during write stalls.

The status quo

- **Observation 1.** ultimately leads to the following options for write stalls.

Slowdowns VS Allowing Write Stalls

- Maintains I/O service at all
- Overall throughput and

Can write stalls be mitigated without sacrificing system resources by leveraging underutilized PCIe and device bandwidth during write stalls?

- Observation 2. reveals an unexploited resource to help mitigate write stalls and increase performance without sacrificing system resources: underutilized PCIe and device bandwidth during write stalls.

Proposed Solution: *KVAccel*

Proposed Solution: ***KVAccel***

- ***KVAccel***'s design is based on two key factors: Disaggregation and Aggregation.

Disaggregation

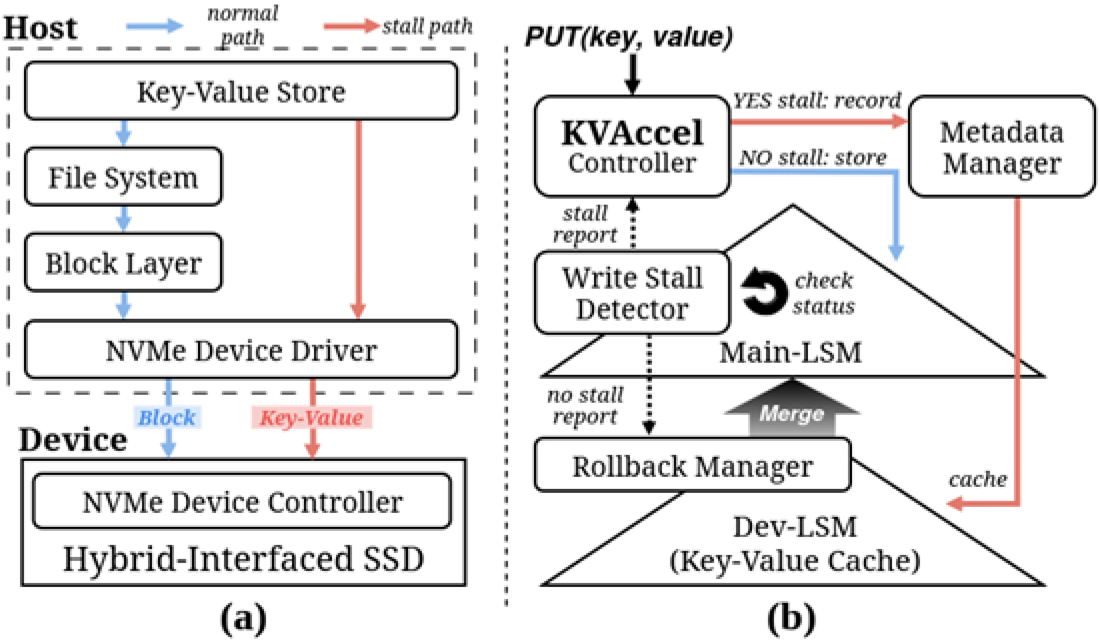
- Division of SSD into hybrid interface (block and key-value) and its required I/O paths
- Maintenance of each interface's separate LSM-Tree

Aggregation

- Manage data from each interface as if it was one database instance
- Unify separate I/O commands and database state with rollback

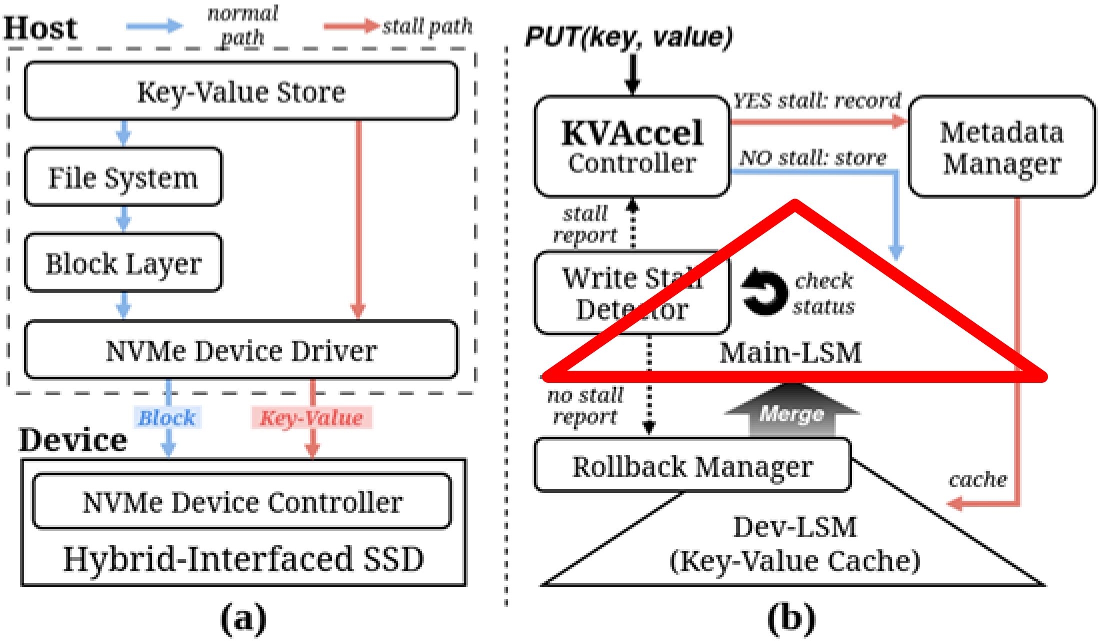
Overview of ***KVAccel***

- Co-Design of Hardware & Software provides 2 I/O paths
- Different I/O paths taken based on the presence of a write stall



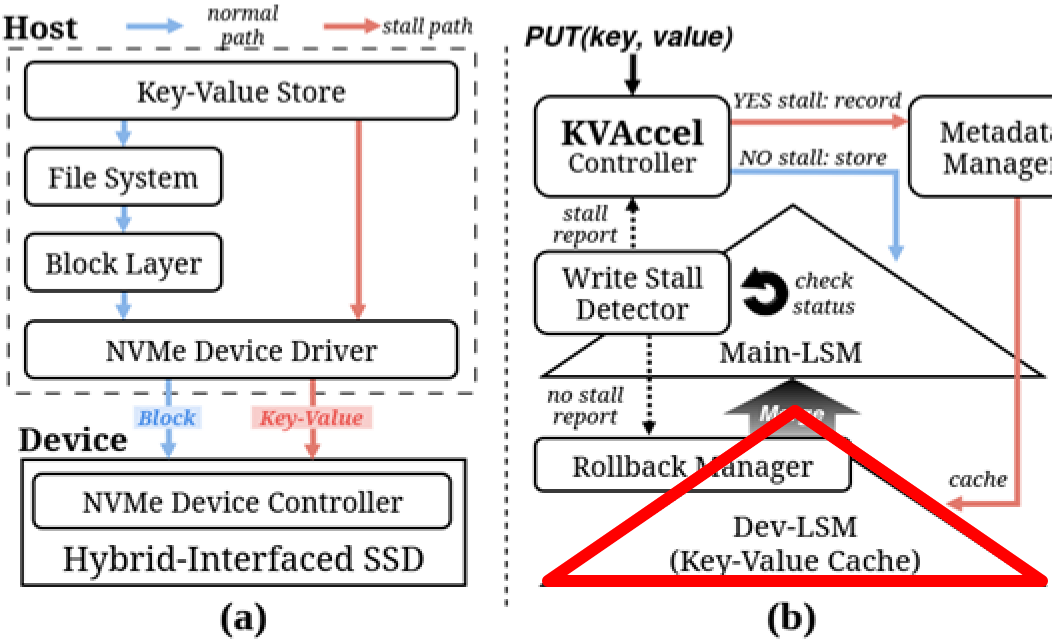
Overview of ***KVAccel***

- Co-Design of Hardware & Software provides 2 I/O paths
- Different I/O paths taken based on the presence of a write stall



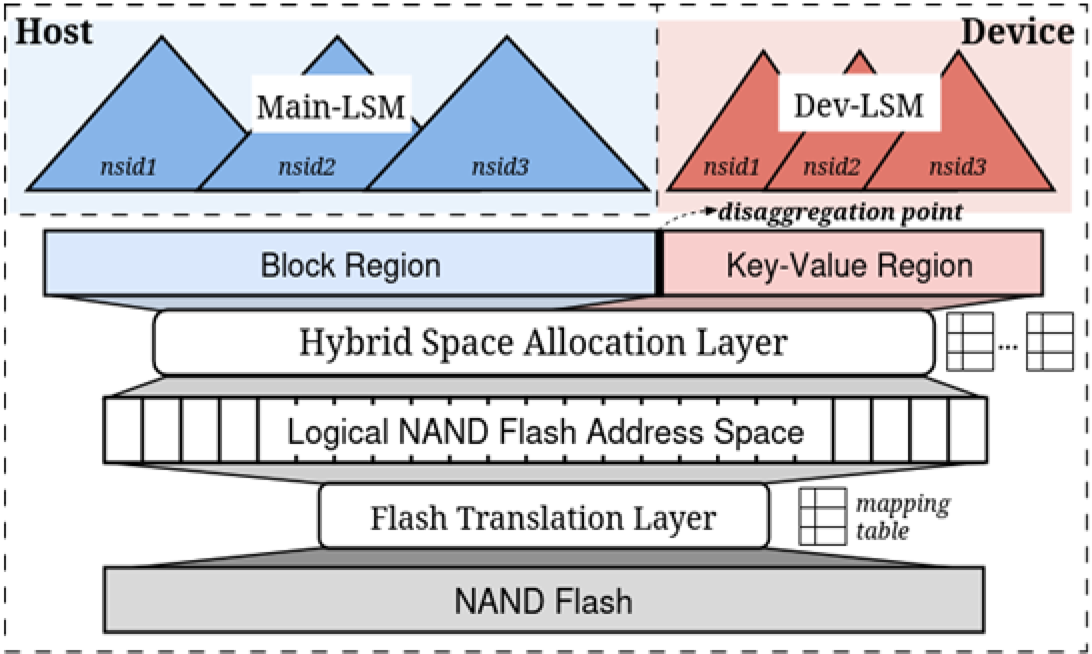
Overview of ***KVAccel***

- Co-Design of Hardware & Software provides 2 I/O paths
- Different I/O paths taken based on the presence of a write stall

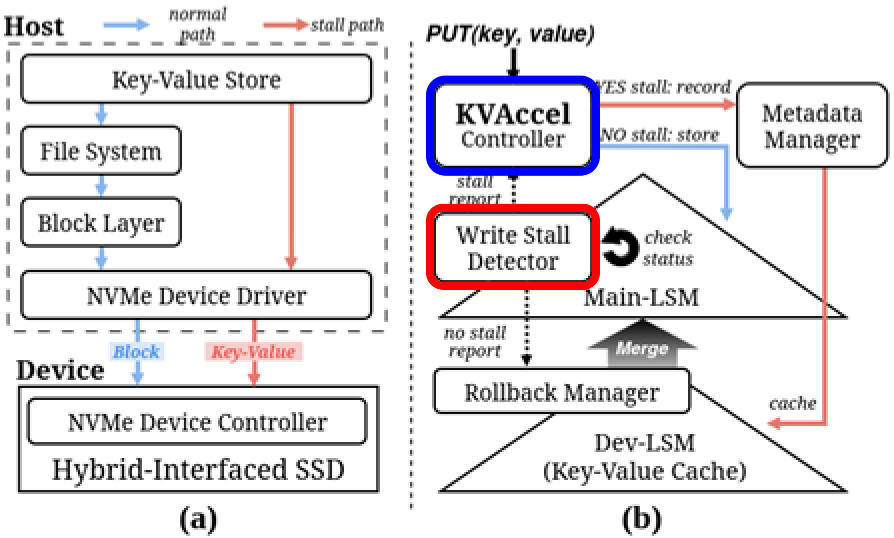


Hybrid Dual-Interface SSD

- Hybrid interface SSD achieved by logical NAND flash address disaggregation via a specified address boundary
  - SSD issues different commands for each interface

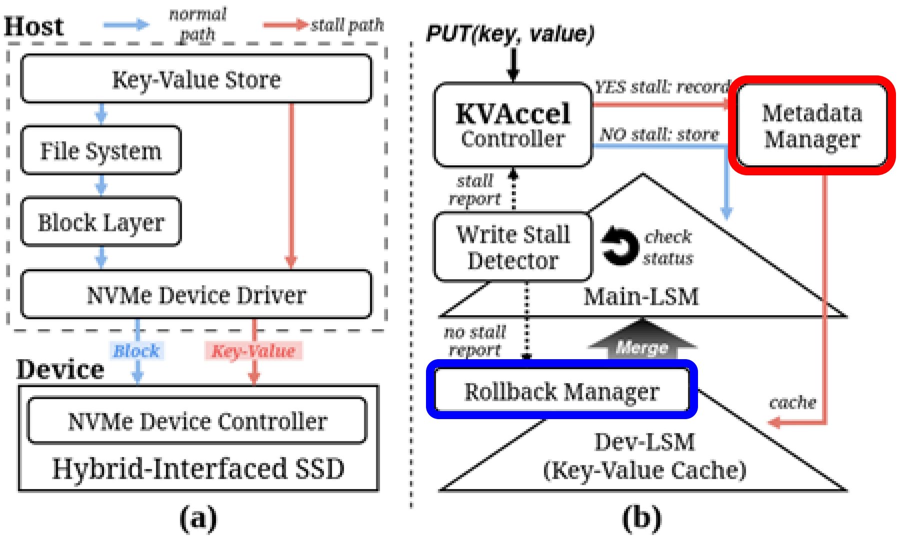


Software Modules(1)



- Detector
  - Detects write stalls checking 3 components
    - # of Level 0 SSTs
    - Memtable size
    - Pending compaction size
- Controller
  - Directs I/O commands to the correct interface based on the Detector's output.

Software Modules(2)



- Metadata Manager
  - Keeps track of KV pairs located in Dev-LSM via a hash table for membership testing
- Rollback Manager
  - Initiates and performs the rollback operation based on the rollback scheduling policy and the Detector’s output.

Rollback Operation: Scheduling

- Rollback refers to return the KV pairs in Dev-LSM back to Main-LSM into one LSM-KVS instance.
- Rollback operation can be scheduled *eagerly* or *lazily* based on workload characteristics.

Eager Rollback

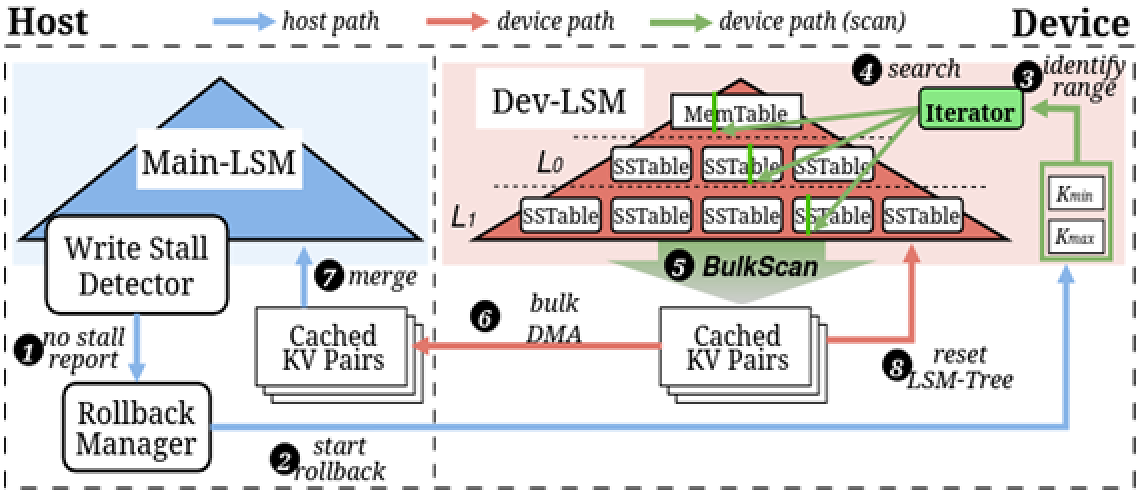
- Perform rollback as soon as there are enough resources available (by using L_0 file count threshold)
- Ideal for a read orientated workload to avoid slow Dev-LSM read operations

Lazy Rollback

- Delay rollback until the current write workload is completely finished
- Ideal for a write intensive workload to lower interference of rollback with write operations

Rollback Operation

- To accelerate rollback, KV pairs are read in bulk using a range scan operation.
- Iterator reads Dev-LSM in its entirety and serializes the KV pairs.
- KV pairs are then sent to the host by performing DMA multiple times.



Evaluation

Evaluation Setup

- Testbed:

KV-SSD on Cosmos+ OpenSSD Platform[7]

TABLE I: Specifications of the OpenSSD platform.

SoC	Xilinx Zynq-7000 with ARM Cortex-A9 Core
NAND Module	1TB, 4 Channel & 8 Way
Interconnect	PCIe Gen2 ×8 End-Points

TABLE II: Specifications of the host system.

CPU	Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz (32 cores), CPU usage limited to 8 cores.
Memory	384GB DDR4
OS	Ubuntu 22.04.4, Linux Kernel 6.6.31

[7]: Cosmos+ OpenSSD Platform: http://www.openssd-project.org/platforms/cosmospl/

LSM-KVS and Benchmark Configurations

TABLE III: LSM-KVS configurations. For all figures, the numbers next to each LSM-KVS refer to compaction thread count. For KVACCEL, the settings refer to the Main-LSM.

LSM-KVS	Compaction Threads (n)	MT Size
KVACCEL(n)	1	128 MB
	2	
	4	
RocksDB(n)	1	
	2	
	4	
ADOC(n)	1	
	2	
	4	

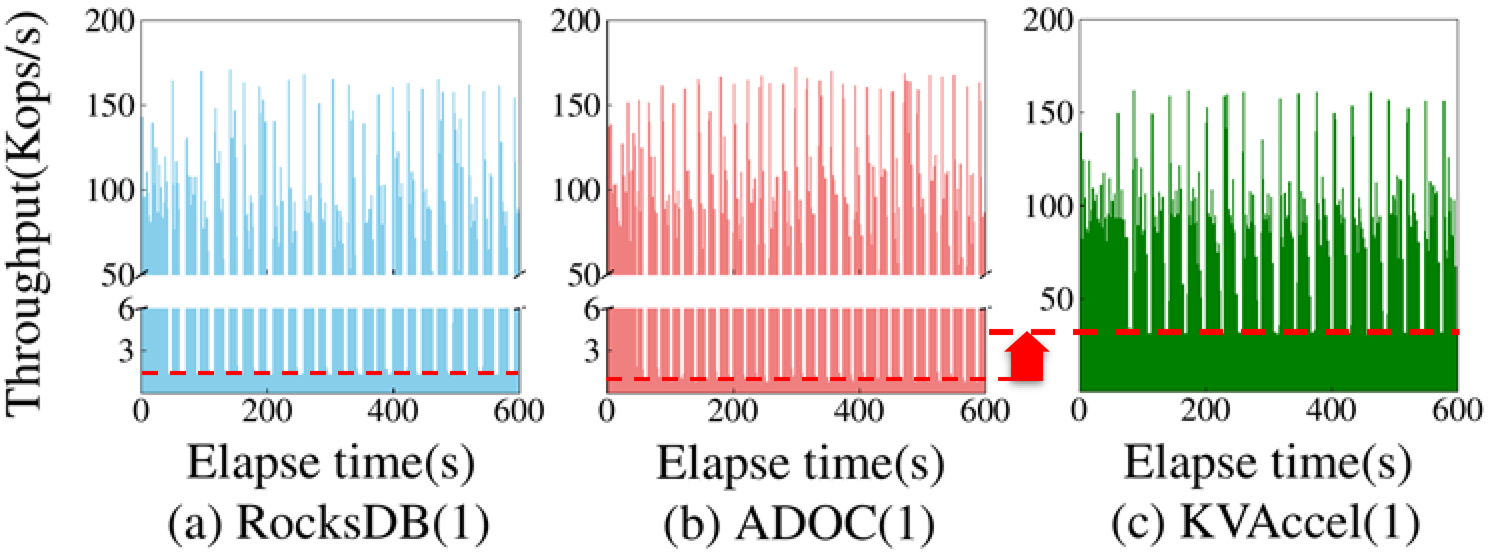
TABLE IV: *db_bench*[8] workload configurations. Each benchmark was run with a 4 B key and 4 KB value size. Workload A,B,C were run for 600 seconds, and Workload D performed 60K read operations.

Name	Type	Characteristics	Notes (write/read ratio)
A	fillrandom	1 write thread	No write limit
B	readwhilewriting	1 write thread + 1 read thread	9:1
C			8:2
D	seekrandom	1 range query thread (Seek + 1024 Next)	Run after initial 20GB fillrandom

[8]: Facebook, "DB Bench" https://github.com/facebook/rocksdb/wiki/

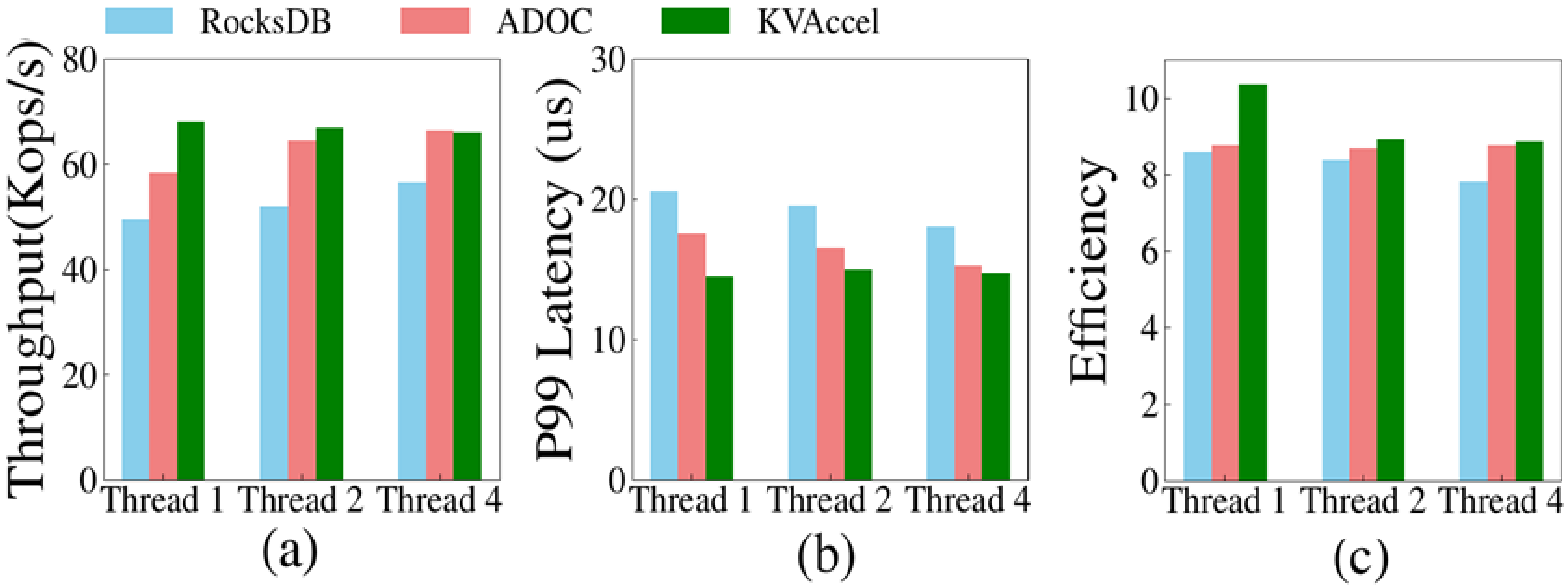
Write Stall Avoidance

- Throughput minimum values greatly increased, as ***KVAccel*** is designed to allow as much throughput as the SSD and system allows without slowdowns.



Performance Evaluation

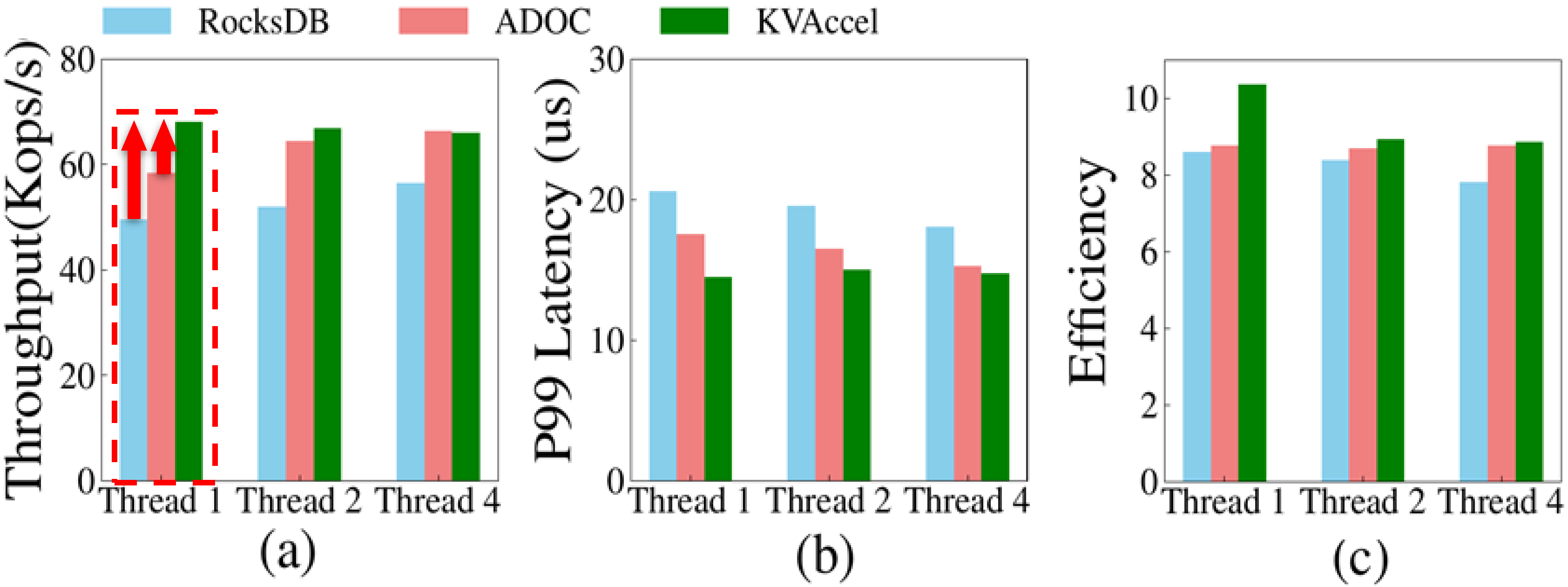
- (a) Throughput, (b) P99 Latency, (c) Efficiency



Performance Evaluation

(a) Throughput

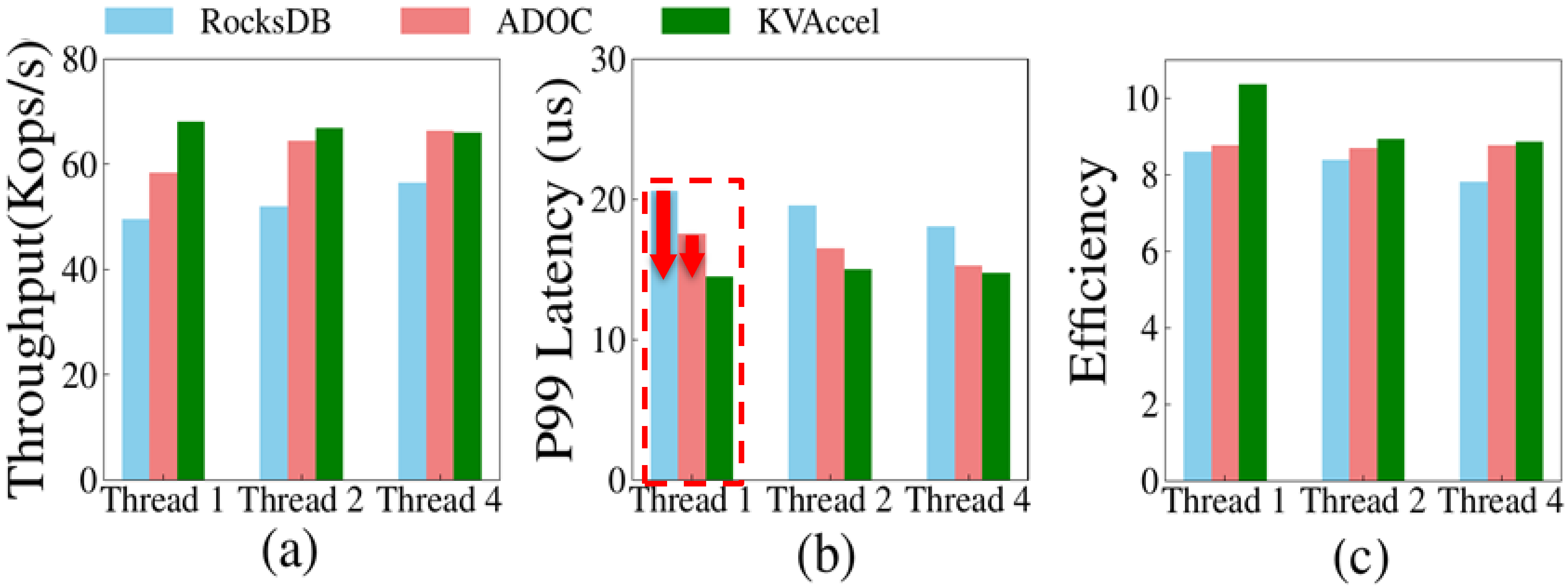
- ***KVAccel*** shows at most a **37%** and **17%** improvement over than RocksDB and ADOC, respectively.



Performance Evaluation

(b) Throughput

- Maximum of **30%** and **20%** decrease in latency was also observed between ***KVAccel*** and RocksDB, ADOC, respectively.

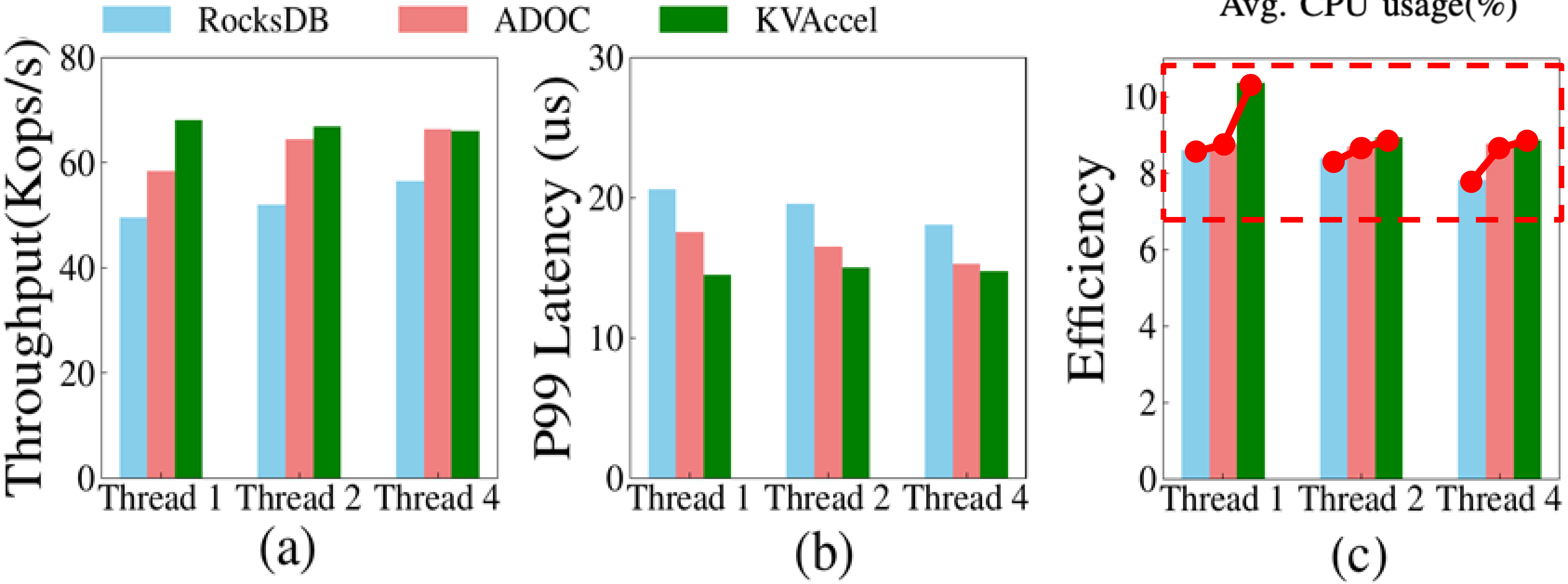


Performance Evaluation

(c) Efficiency

- ***KVAccel*** maintains the better efficiencies in host machine's resources between all LSM-KVS compared.

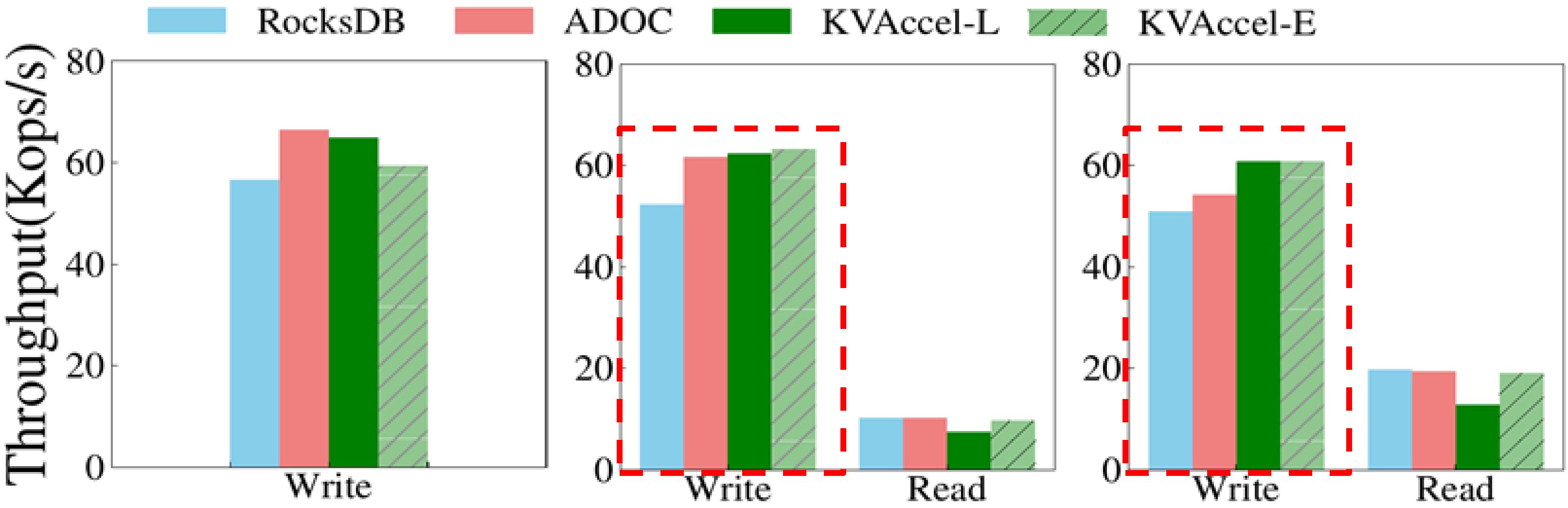
$$\text{Efficiency} = \frac{\text{Avg. Throughput(MB/s)}}{\text{Avg. CPU usage(\%)}}$$



Rollback Policies Evaluation

Eager vs Lazy Rollback analysis

- From (b) and (c), we observe that it still outperforms RocksDB and ADOC under read-oriented workloads



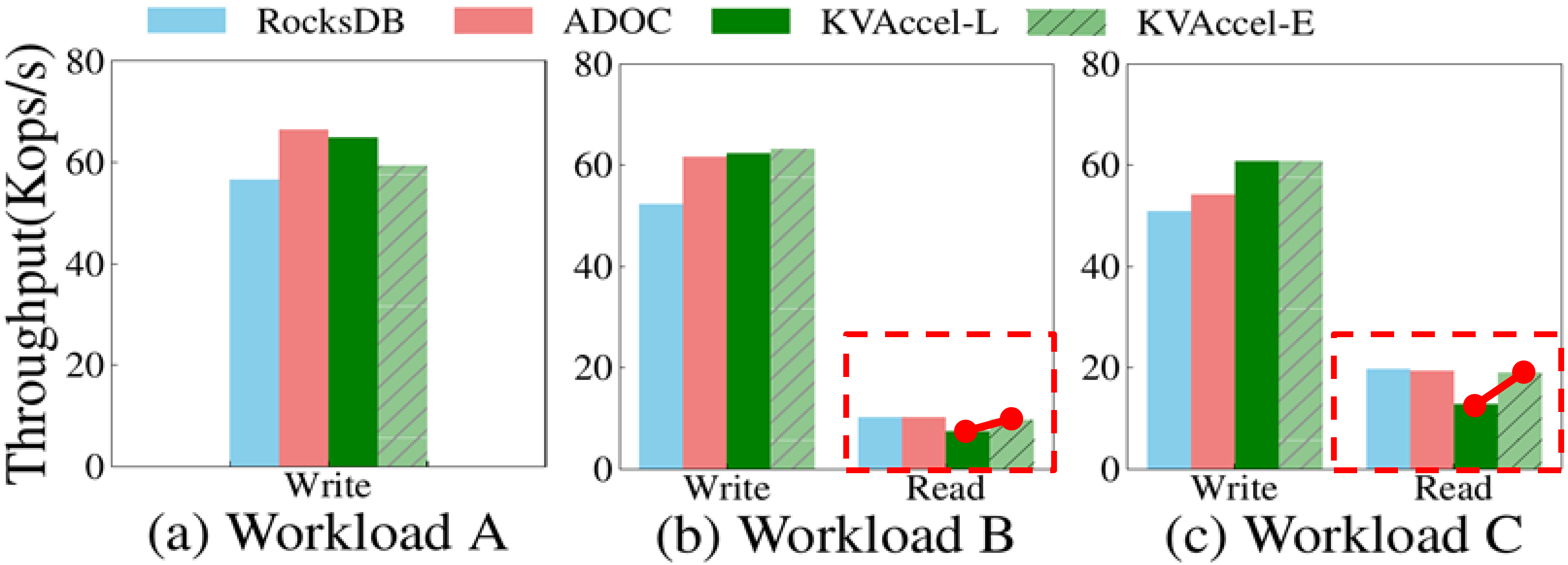
(a) Workload A (b) Workload B (c) Workload C

W:R=10:0 **W:R=9:1** **W:R=8:2**

Rollback Policies Evaluation

Eager vs Lazy Rollback analysis

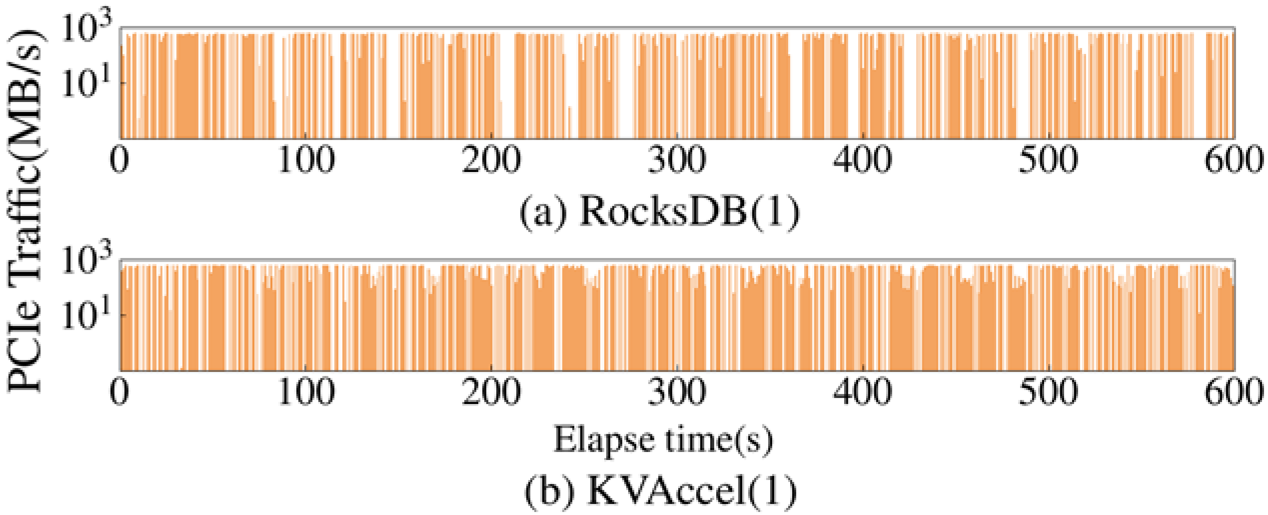
- As the read ratio increases, Eager Rollback becomes increasingly advantageous



W:R=10:0 **W:R=9:1** **W:R=8:2**

PCIe Traffic Usage

- More available PCIe traffic exploited
- ***KVAccel*** takes advantage of its dual interface and demonstrate higher PCIe utilization over RocksDB.



Conclusion

Conclusion

- Prior work addresses write stalls to a limited extent
    - Hardware and software are treated in isolation

- ***KVAccel*** achieved a 17% improvement in throughput and a 20% reduction in latency compared to ADOC.

- ***KVAccel*** demonstrates the effectiveness of hardware-software co-design
    - Alleviates write stalls by utilizing:
        - Under-used PCIe bandwidth
        - Computational capabilities within SSDs

Thank you!

<Camera-ready paper>
Can be found on Google Scholar


KVACCEL: A Novel Write Accelerator for LSM-Tree-Based KV Stores with Host-SSD Collaboration

- Contact
  - Kihwan Kim / lewis461@sogang.ac.kr
  - Hyunsun Chung / hchung1652@sogang.ac.kr
  - Seonghoon Ahn / ok10p@sogang.ac.kr
  - Data-Intensive Computing & AI Systems Laboratory https://discos.sogang.ac.kr/