HiDPU: A DPU-Oriented Hybrid Indexing Scheme for Disaggregated Storage Systems

Wenbin Zhu[1], Zhaoyan Shen[1,*], Qian Wei[1], Renhai Chen[2,3,*], Xin Yao[3], Dongxiao Yu[1], Zili Shao[4]

[1]Shandong University

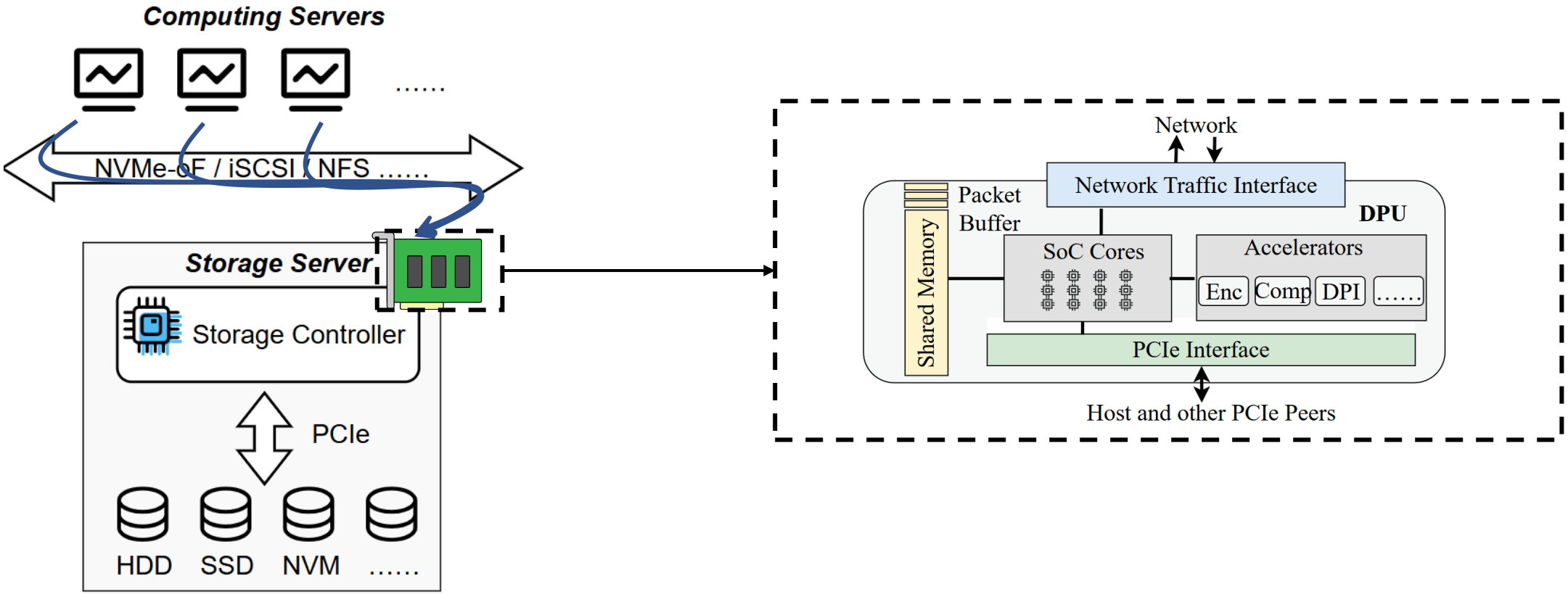
[2]Tianjin University

[3]Huawei Technologies Co., Ltd

[4]The Chinese University of Hong Kong

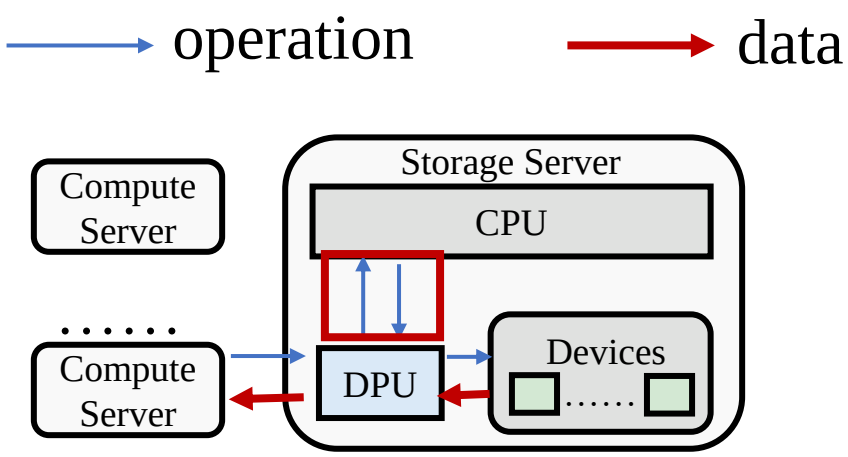
Background and Problem

- **Data Processing Unit (DPU)** for Disaggregated Storage
  - DPU has been proposed to replace the Network Interface Card (NIC) for data access acceleration
    - Computing, Networking and Storage capability



Background and Problem

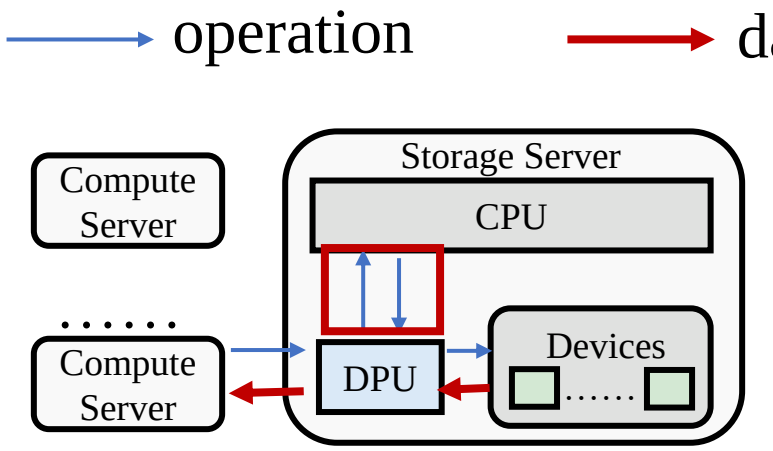
- **Data Processing Unit (DPU)** for Disaggregated Storage
  - DPU has been proposed to replace the Network Interface Card (NIC) for data access acceleration
    - Computing, Networking and Storage capability
    - Directly access PCIe devices, without data copy between CPU memory

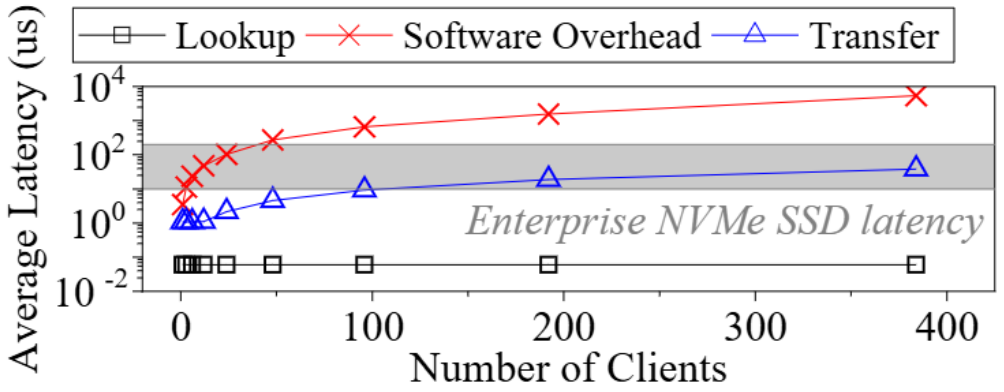


Still involves CPU

Background and Problem

- **Data Processing Unit (DPU)** for Disaggregated Storage
  - DPU has been proposed to replace the Network Interface Card (NIC) for data access acceleration
    - Computing, Networking and Storage capability
    - Directly access PCIe devices, without data copy between CPU memory
- **CPU-centric Address Translation**
  - Limited CPU resources in Storage Server + Overhead of involving CPU ***VS*** Massive concurrent requests





Still involves CPU

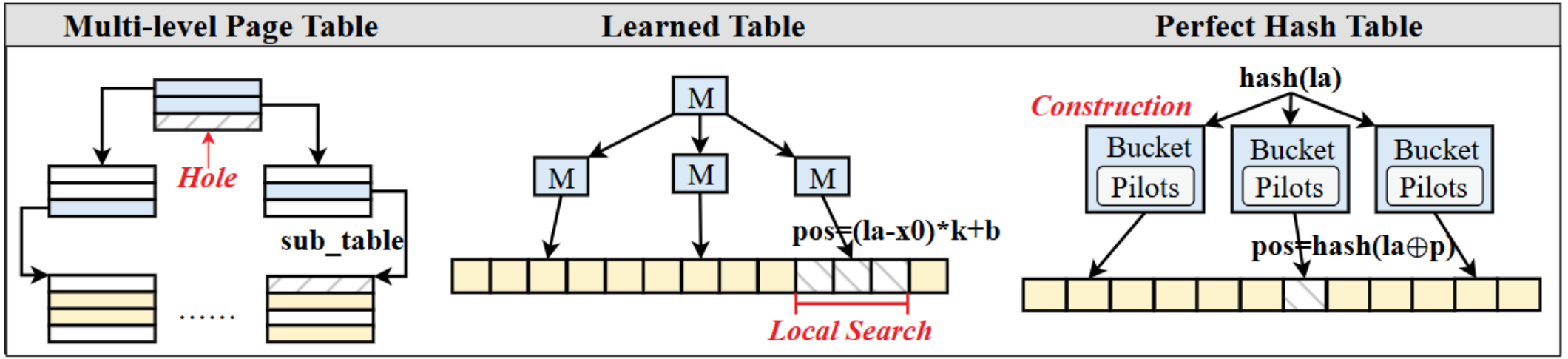
CPU-centric address translation incurs high latency

How about to do address translation in DPU? 🤔

Challenges and Current Indexes

- Challenges for indexing in DPU
  - Limited memory resources
  - High interaction overhead
  - Weak computational power

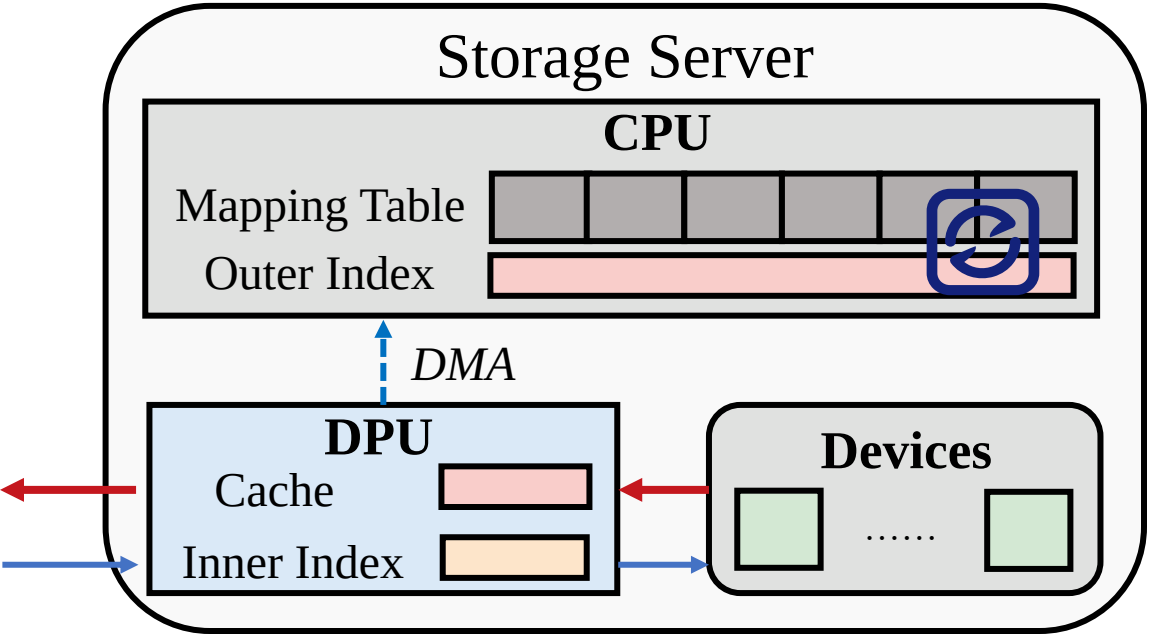
Current Indexes	Pros	Cros
Multi-level Page Table	Lookup	Space
Learned Table	Space	Lookup
Perfect Hash Table	Lookup & Space	Update



Currently, there is no index structure that effectively address these challenges.

HiDPU

- Our solution
  - **HiDPU: DPU-Oriented Hybrid Indexing schema**



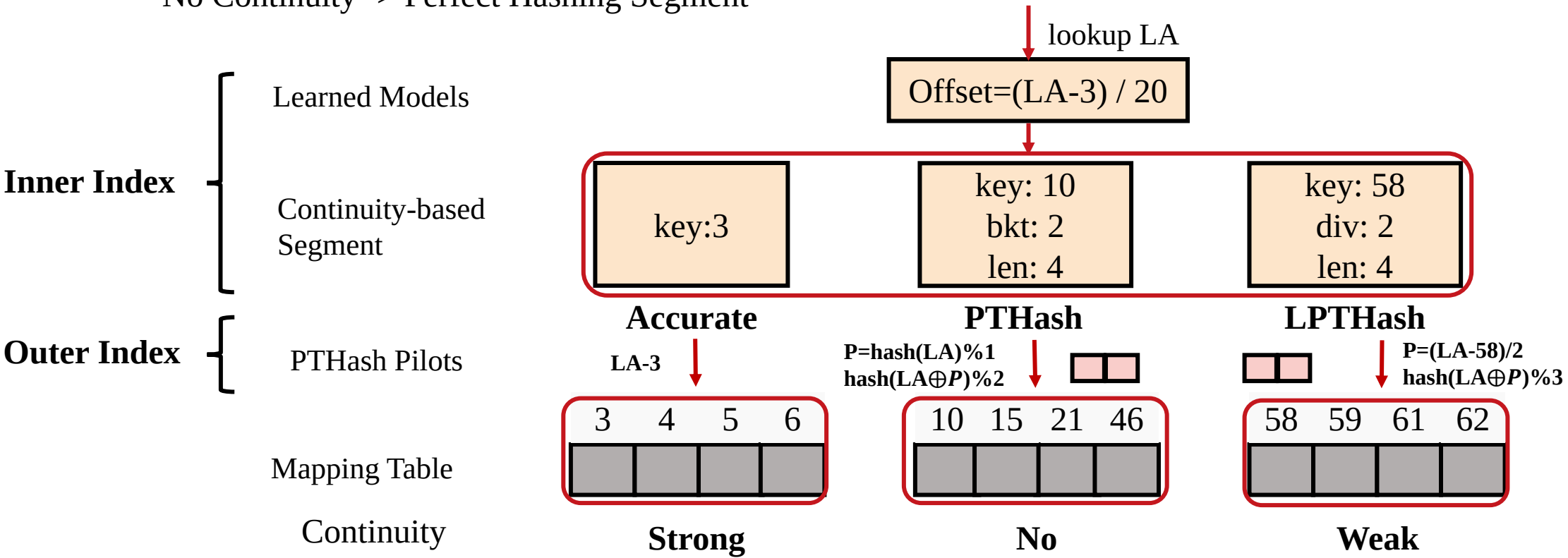
- Hybrid Index
  - Inner index: learned index, continuity-based segments
  - Outer index: second-level hash
  - Leaf: Mapping Table
- DPU-Offloaded Cache
  - Reduce extra DMAs
- Asynchronous Index Updates
  - Query in DPU
  - Update in CPU

HiDPU

- **Hybrid Index**
  - Learned Index
  - Continuity-Based Segments
    - Strong Continuity -> Accurate Segment
    - Weak Continuity -> Linear-based Perfect Hashing Segment
    - No Continuity -> Perfect Hashing Segment

Approximate search but space efficient

Deterministic search and space efficient



HiDPU

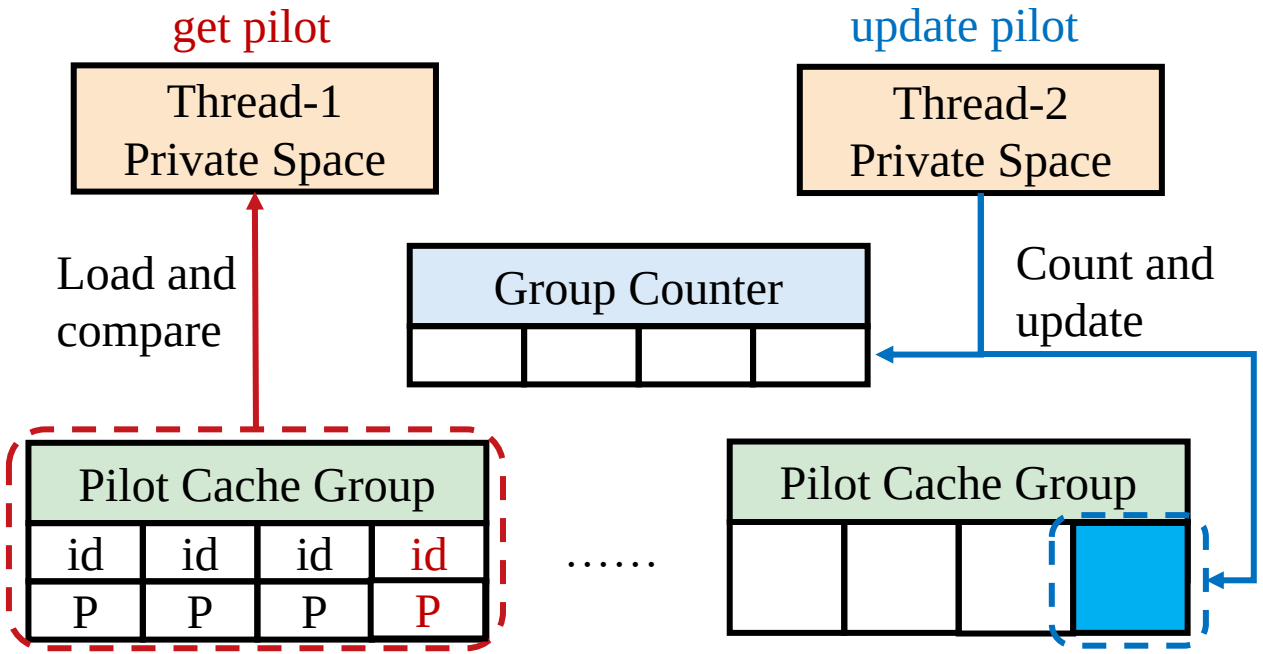
- **DPU-Offloaded Pilot Cache**
  - Cache Pilots in Shared Memory, saving extra DMA for PTHash-based Segments

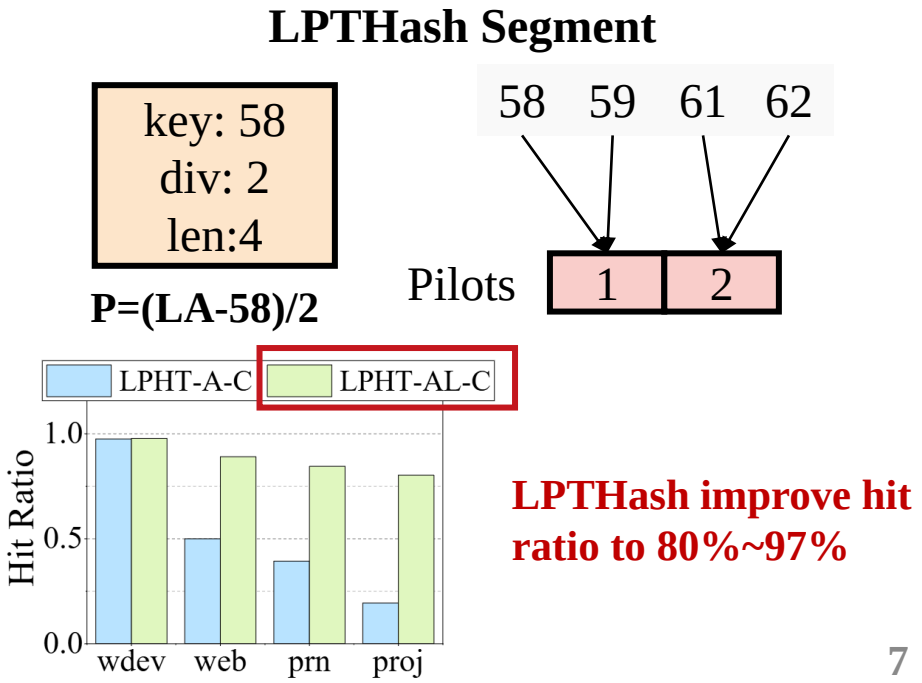
  Problem-1: how to handle hundreds of threads concurrently access cache?

  Optimistic Locking + Atomic Counter

  Problem-2: how to guarantee cache hit ratio?

  Linear-Based PTHash

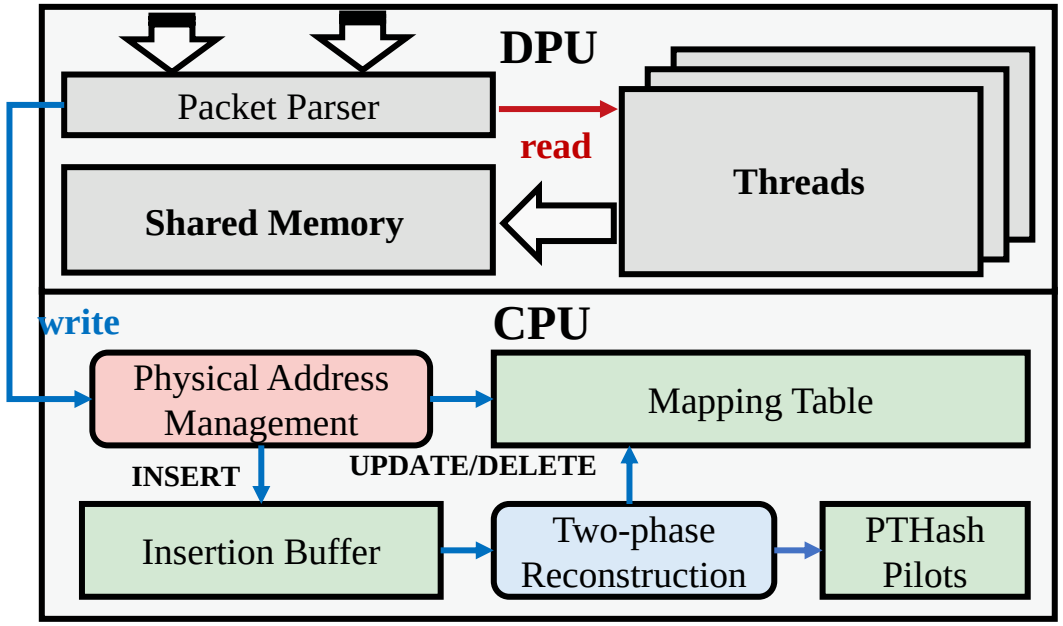


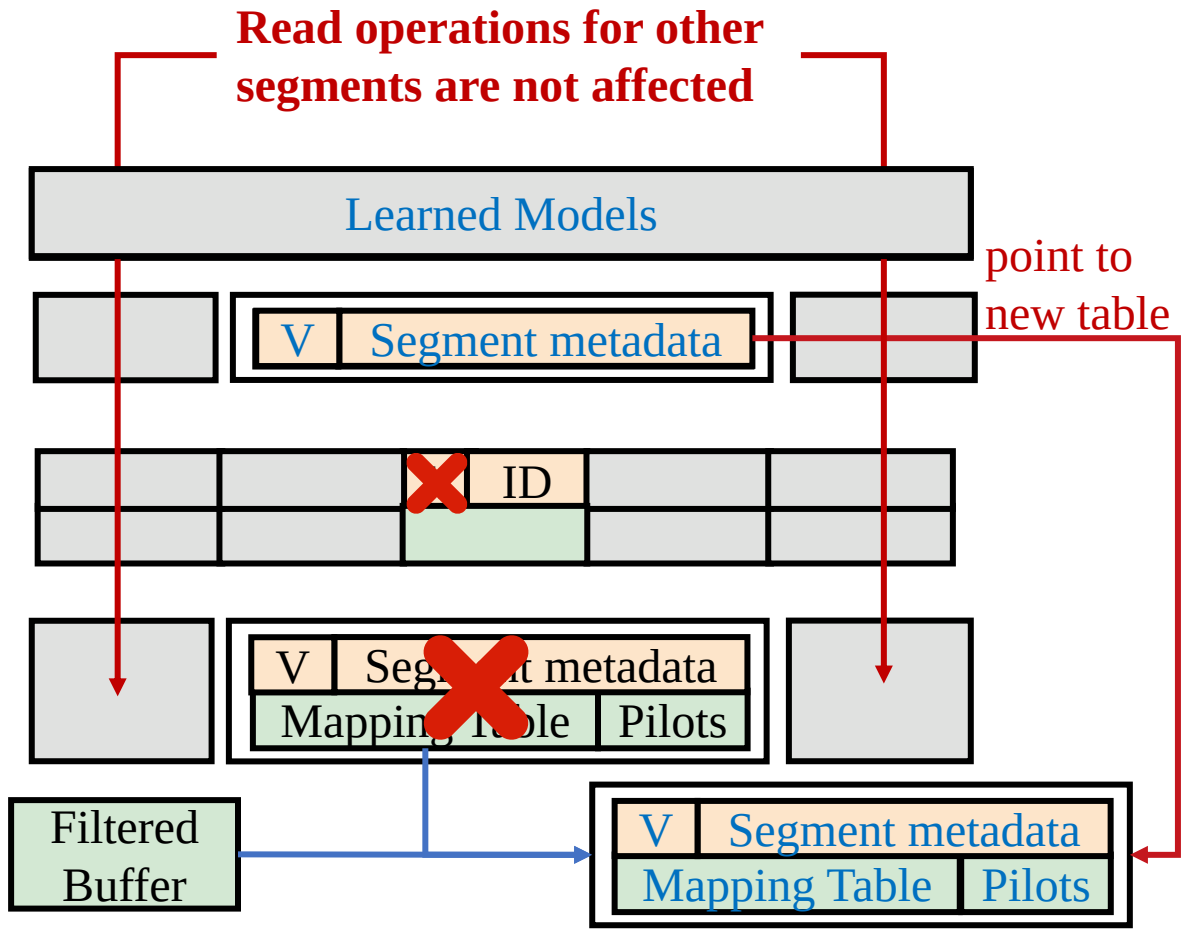


LPTHash improve hit ratio to 80%~97%

HiDPU

- **Asynchronous Index Updates**
  - Index Asynchronous: DPU——simple and frequent Query, CPU——complex and periodical Update
  - Operations: UPDATE, DELETE and INSERT
  - Cuckoo hash-based insertion buffer
  - Two-phase Reconstruction
    - Buffer fill up -> Local Reconstruction
    - System idle -> Global Reconstruction





Evaluations

- Testbeds
  - 96-core 2.4GHz Intel Xeon Platinum 8260 CPU, 512GB of DDR4 DRAM
  - Huawei Hi1823 DPU
    - 4MB shared DPU memory and 4KB isolated SPRAM for each physical thread.
  - Optimizations: Fixed-Point Model Prediction, Pipeline-based Local Search, and Data Reuse in private memory

A functional **CPU-only HiDPU simulator** can be found at: https://github.com/quieoo/lmpthash.

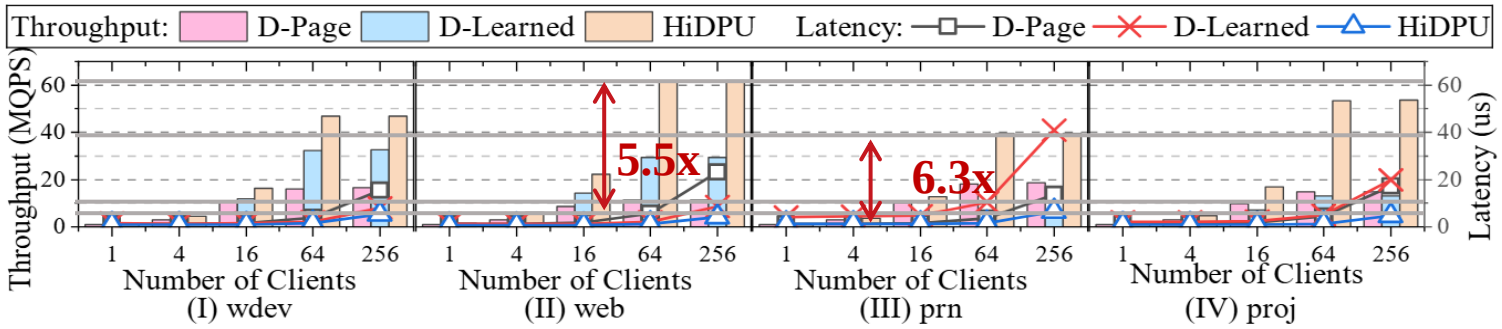
- Baselines
  - D-Page: Three-level page table, with the first level offloaded to the DPU
  - D-Learned: Builds a learned index (PGM-index[1]) on the mapping table, adjusting the error bound to ensure the index can be offloaded to the DPU. All implementation optimizations of HiDPU are also applied.
- Workloads
  - Micro I/O traces: Block I/O trace from Microsoft Research Cambridge([2])
  - Macro I/O traces: Captured from a FEMU server running various workloads ([3])

	Micro I/O traces				**Macro I/O traces**			
Name	wdev	web	prn	proj	oltp	websrv	tpcds	ycsbc
Unique addresses (million)	0.1	17	21	180	20	4	15	23
Total accesses (million)	2	69	55	202	47	48	203	346

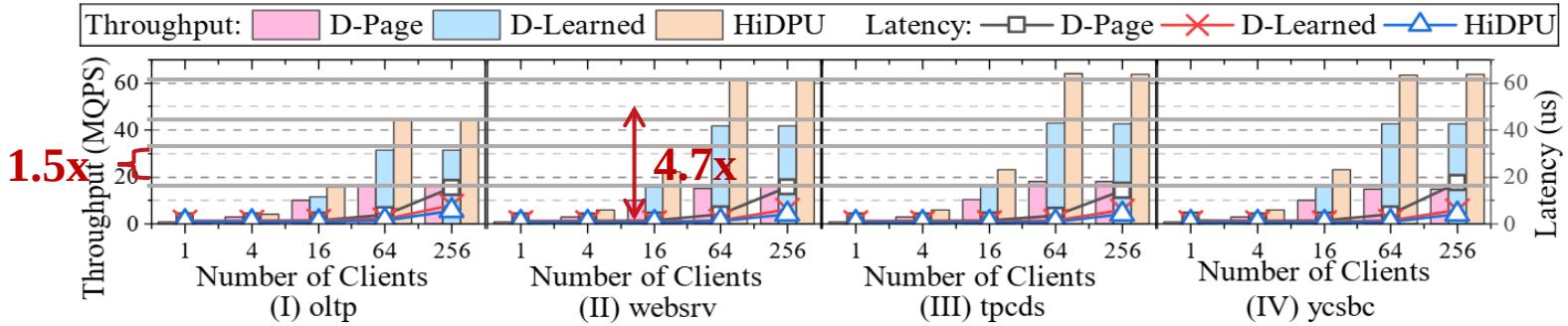
[1]: Ferragina P, Vinciguerra G. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds.
[2]: SNIA. Block i/o trace: Microsoft research cambridge - msr cambridgeblock i/o traces. http://iotta.snia.org/traces/block-io/388.
[3]: https://github.com/quieoo/FEMU_Trace

Evaluations

- Overall Performance
  - D-Page suffers from multiple DMAs
  - D-Learned performs better in real-world scenarios, but its performance varies significantly with address continuity.
  - HiDPU outperforms all workloads (with **92%** probability saving one DMA)
    - up to **5.5x** with D-Page and **6.3x** with D-Learned on Block IO trace, **4.7x** with D-Page and **1.5x** with D-Learned on real-world scenarios.



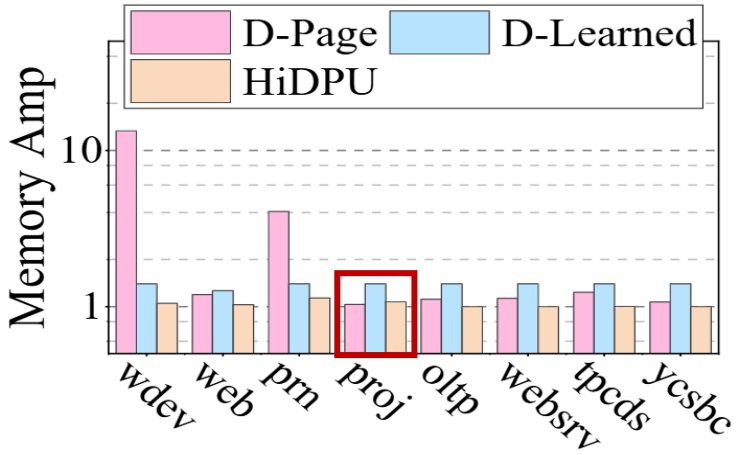
(a) MRC Block I/O trace

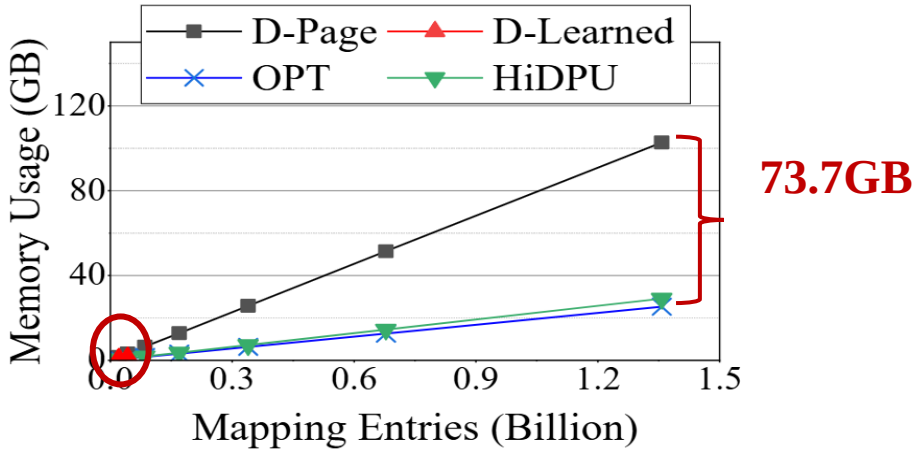


(b) Real-world benchmark scenarios

Evaluations

- Memory Consumption
  - Memory amplification compared to the Optimal Case (only storing mapping entries):
  - D-Page : “holes” in each level table (1.06X ~ 13.3X)
  - D-Learned: storing logical addresses in mapping entry for Local Search (8 bytes per entry, 1.4X)
  - HiDPU: Perfect Hash Table overheads (1.02X ~ 1.13X)
- Petabyte-scale
  - D-Page incurs high Host memory overhead
  - D-Learned fails to build a learned index that fits within the DPU memory.
  - **HiDPU efficiently handles petabyte-scale storage mapping indexing with constrained DPU memory and minimal host memory overhead**

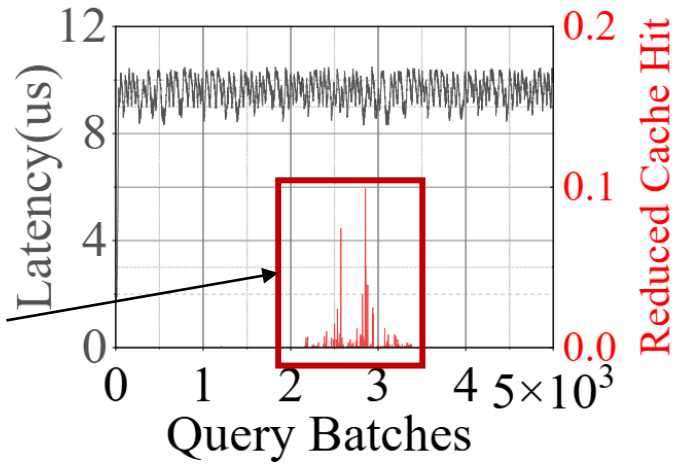


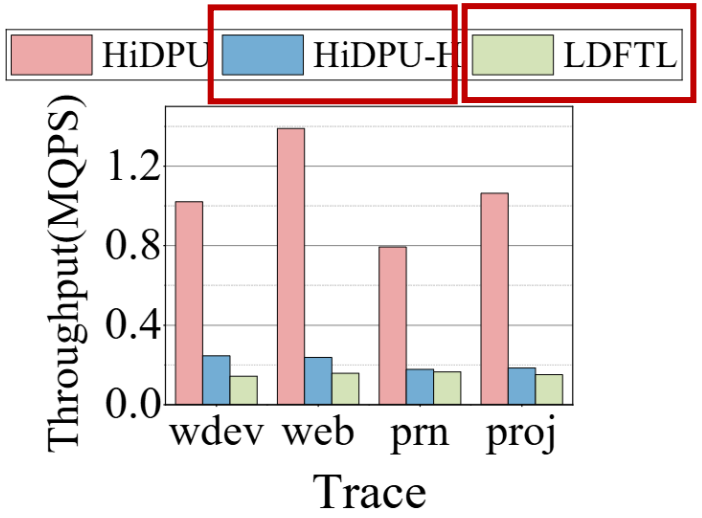


Evaluations

- Index Reconstruction
  - Local Reconstruction causes Pilot Cache updates and miss (~10%)
  - Minimum effects on read performance (~0.69%)
- Compare with State-of-the-Art
  - LearnedFTL([1]): cached mapping table (CMT) + learned index (with a bitmap indicts accurate)
    - Failed to be offloaded into DPU

Local Reconstruction happens





[1]:Wang S, Lin Z, Wu S, et al. LearnedFTL: A Learning-Based Page-Level FTL for Reducing Double Reads in Flash-Based SSDs

Conclusion

- The DPU has been employed to accelerate the data path in disaggregated storage.
- Without indexing, the DPU still relies on CPU for address translation, adding extra overhead
- We propose **HiDPU**, a solution to offload address translation in DPU
- HiDPU integrates techniques that address challenges: limited space, interaction overhead and limited computation
- We have implemented and evaluated HiDPU prototype on a real DPU

Thanks!

Q&A

wenbinzhu@mail.sdu.edu.cn