Towards Building a High-Performance, Scale-In Key-Value Storage System

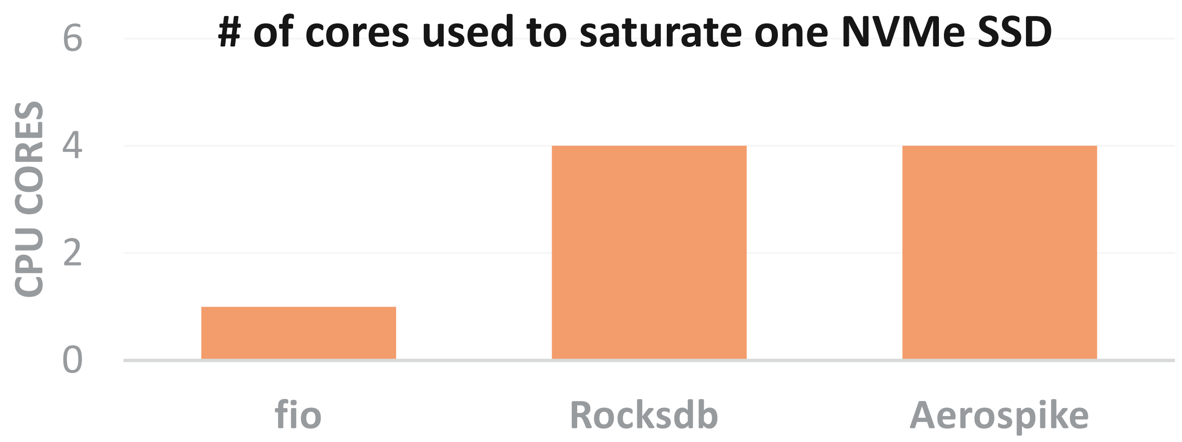
Yangwook Kang, Rekha Pitchumani, Pratik Mishra, Yang-suk Kee, Francisco Londono, Sangyoon Oh, Jongyeol Lee, and Daniel D. G. Lee

Samsung Electronics

Challenges in Leveraging Fast SSDs

We observed that:

- Increasing IO bandwidth of SSDs requires more host system resources
  - At least, one dedicated IO core is needed to saturate one NVMe SSD *without any interference*



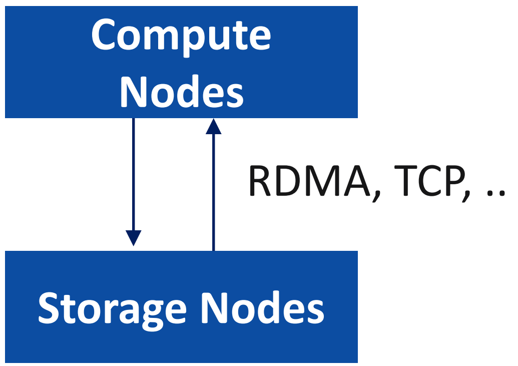
- There are already many compute and IO intensive tasks in enterprise storage systems
  - Indexing, journaling, compressions, deduplications ..
  - Fast PCI-e based SSDs are making the situation worse

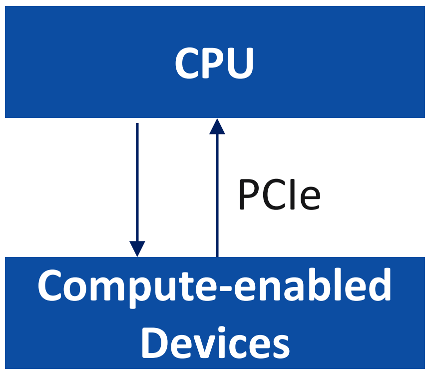
Challenges in Leveraging Fast SSDs

- Increased per-device resource demands can limit scalability or performance
  - CPU and memory are limited resources
  - Only a small number of devices in a node can be supported at their full performance

- Offloading resource-intensive tasks can be helpful
  - Compute and storage nodes (offload IO tasks to remote storage nodes)
  - Local compute-enabled devices (GPU, Smart NIC …)

+ *Separate CPUs for IOs*
− *Network congestion*
− *Lots of data transfer*

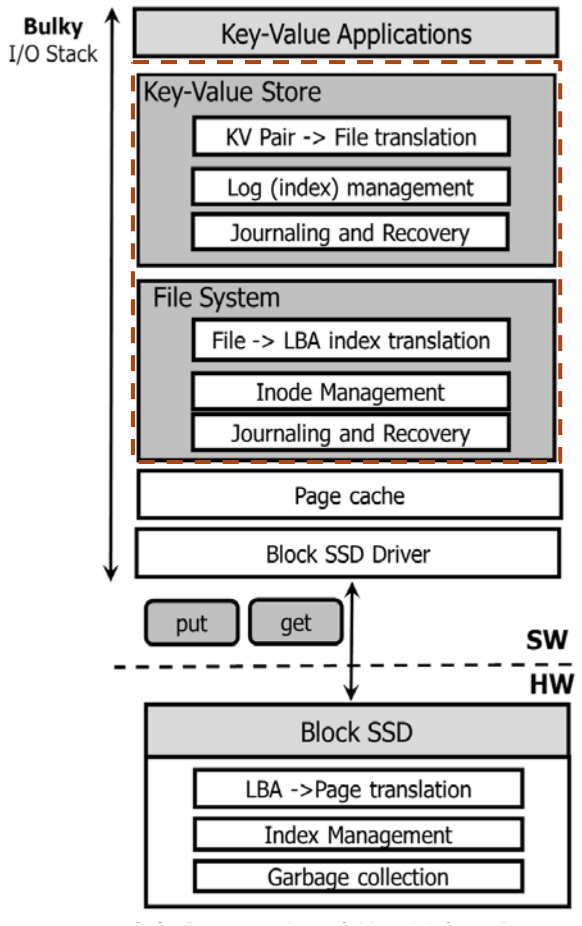




+ *Local data processing*
+ *Efficient data movement*

What can be offloaded from Host CPUs?

- Many resource-intensive tasks are running in storage systems
  - e.g. networking, checksum, compression, erasure coding, indexing ..
- Key-Value Stores
  - Widely used as an internal data store in scale-out storage systems
    - Data can be independently moved to other nodes or devices
  - Complex storage stack that requires lots of host CPU and memory
    - indexing, logging and data reorganization
  - Improving its efficiency can benefit the systems running on top of it

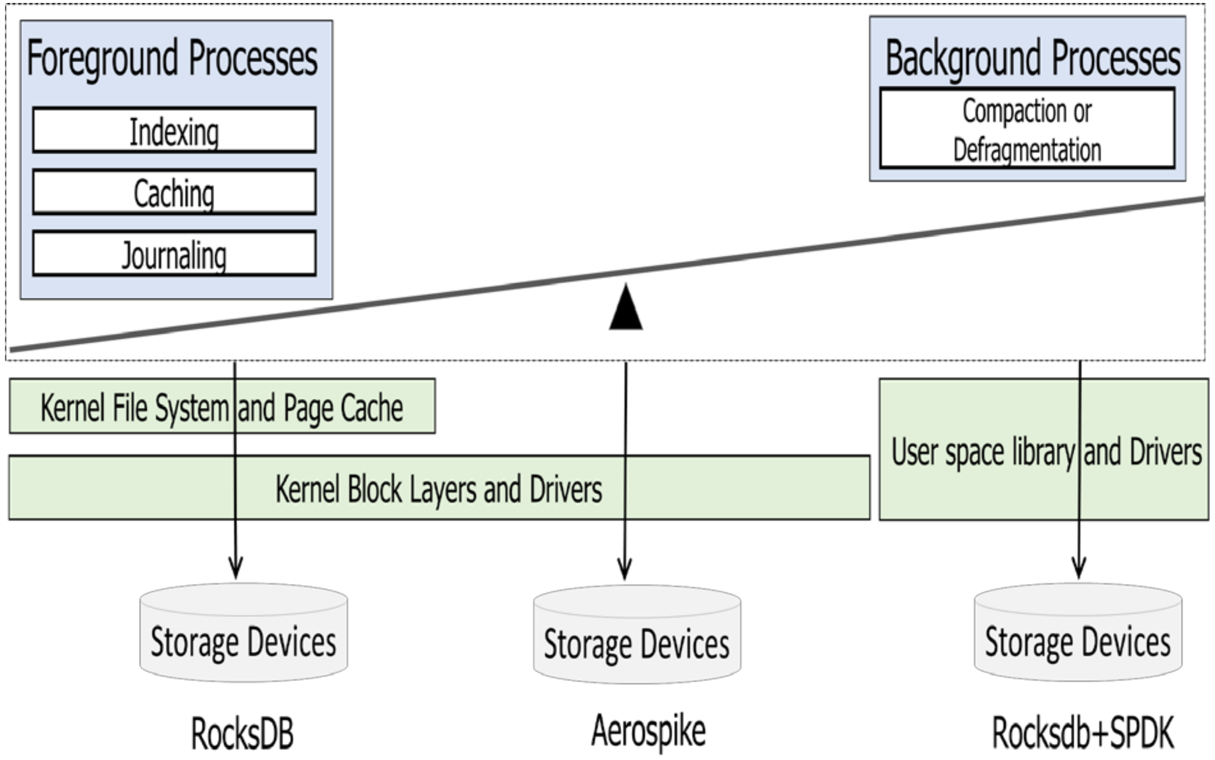


Conventional Key-Value Stores

Existing Approaches In Conventional Key-Value Stores

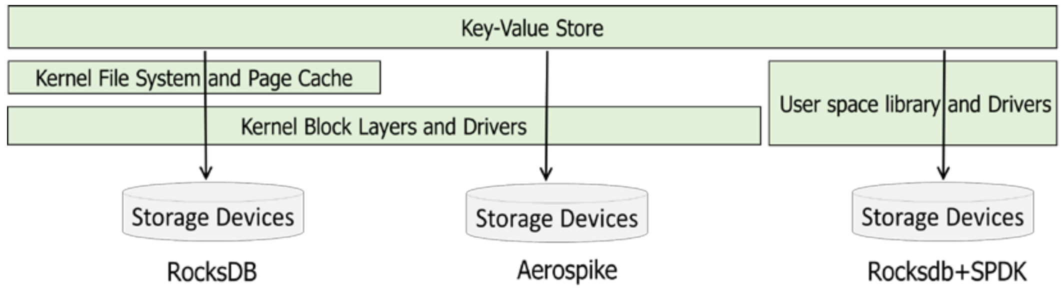
- **Offloading tasks to background processes**
  - Foreground logging:
    to accept the requests at the speed of devices
  - Background organization:
    re-organize the written data

- **Bypassing kernel layers**
  - Directly access block device through user-level or kernel-level drivers

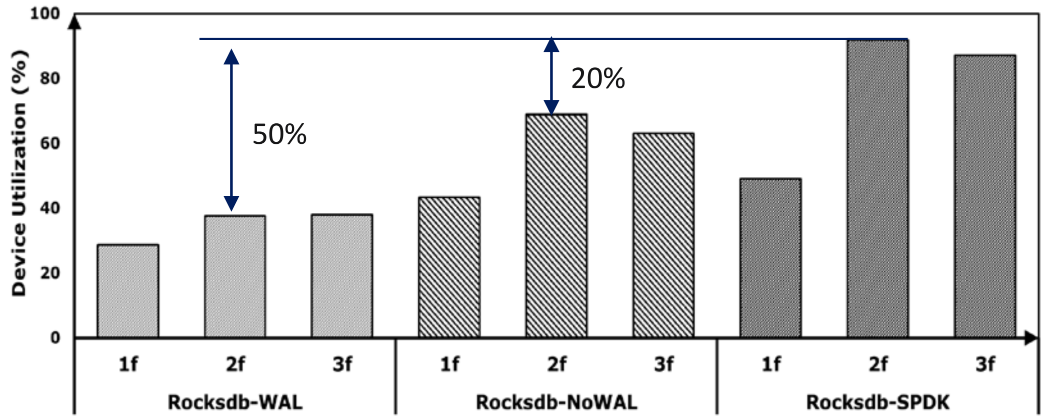


Performance Impact of I/O Stacks

- **Use of Direct IO (Aerospike)**
  - Bypass page cache & kernel file system
  - Did not find much difference in performance compared to RocksDB



- **RocksDB-SPDK**
  - Provides 2x better resource utilization and performance than RocksDB
    - WAL is disabled
    - Without WAL, Rocksdb's performance can also be improved
  - Compared to Rocksdb-NOWAL, Rocksdb-SPDK provides 20% better performance
    - Asynchronous I/O, Huge pages, large block sizes
    - 20% improvement on IO efficiency was not enough to solve the issues with limited scalability

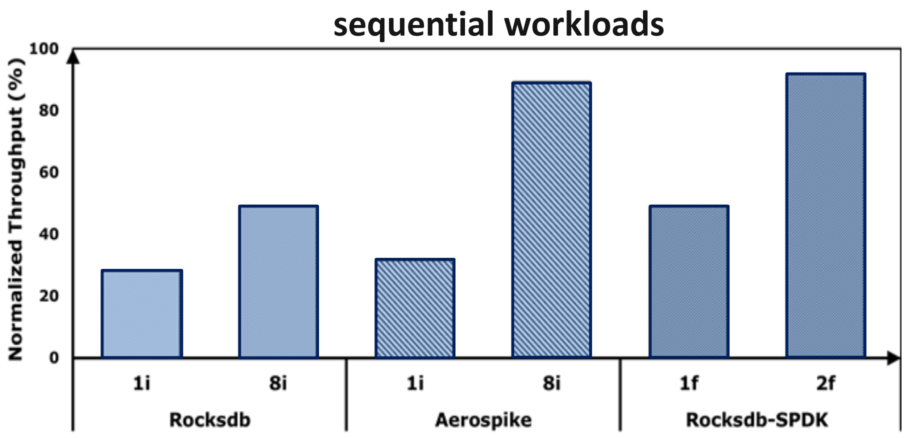


COLLABORATE. INNOVATE. GROW.

SAMSUNG

Resource Demands of Foreground and Background Processes in Host Key-Value Stores

- **Overheads of Foreground Processes**
  - Rocksdb and Aerospike require 8 flush threads
    - Rocksdb requires Write-ahead Log (WAL) and *fsync*
    - Aerospike uses a pool of synchronous I/O threads
  - Rocksdb-SPDK saturates the device with 2 flush threads



Resource contention problem still exists

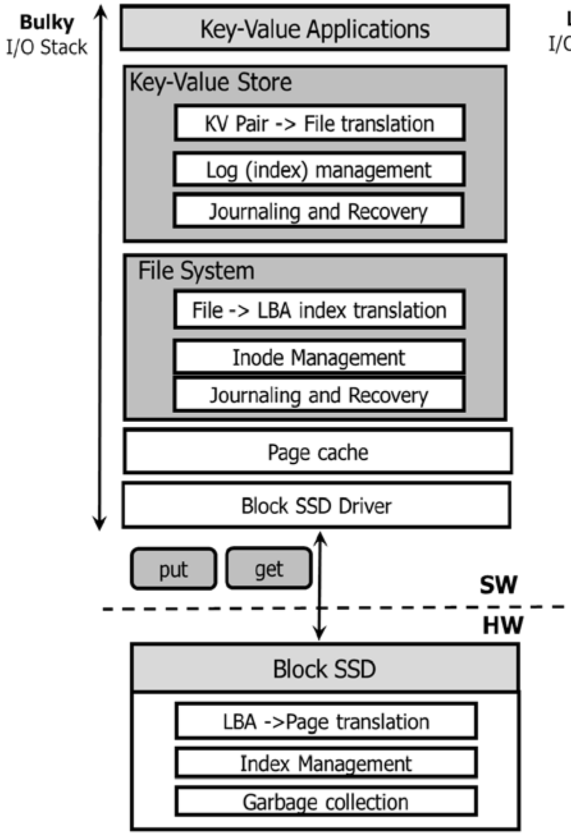
- **Overheads of Background Processes**
  - The amount of overheads depends on several factors
    - the number of key-value pairs, the size of values, and the type of a workload
    - Many overwrites or randomly generated keys increases the overheads
  - Slow background processes can make foreground processes stall or slow-down
  - Overall performance degradation was around 20% - 80% compared to the sequential performance

Offloading the Key-Value Management to Storage Devices

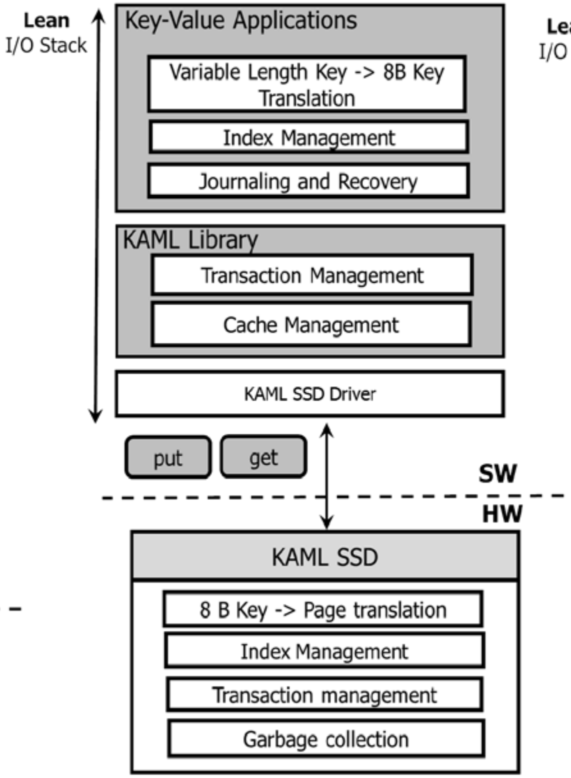
- Expected benefits of offloading
  - Host foreground processes -> save 1-7 flush threads per device
  - Host background processes -> save 2-3 background threads per device

- Among various compute resources that are available today, we chose to use SSDs
  - No extra data transfers for key-value processing
  - Use of existing hardware components
    - SSDs are already capable of supporting fixed-length key and variable-length value requests
  - Avoid metadata update overheads
    - no indexing, journaling, WAL in a host system

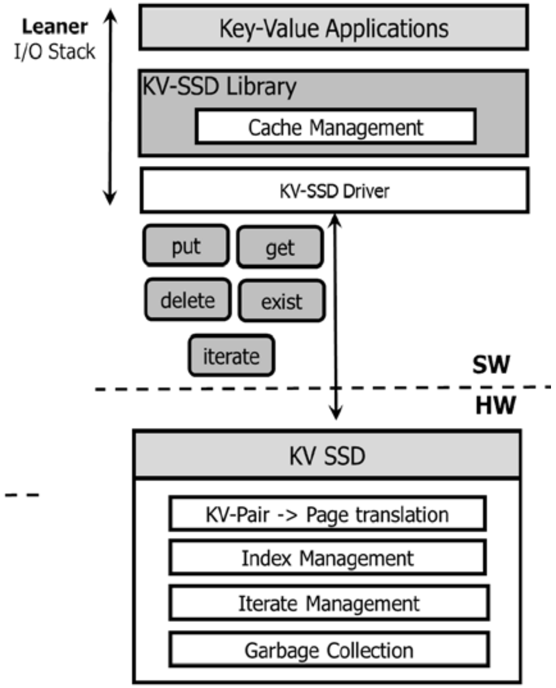
Finding a boundary between Key-Value SSD and Host for Performance and Scalability



Conventional Key-Value Store



KAML (Jin et. al.)



KVSSD

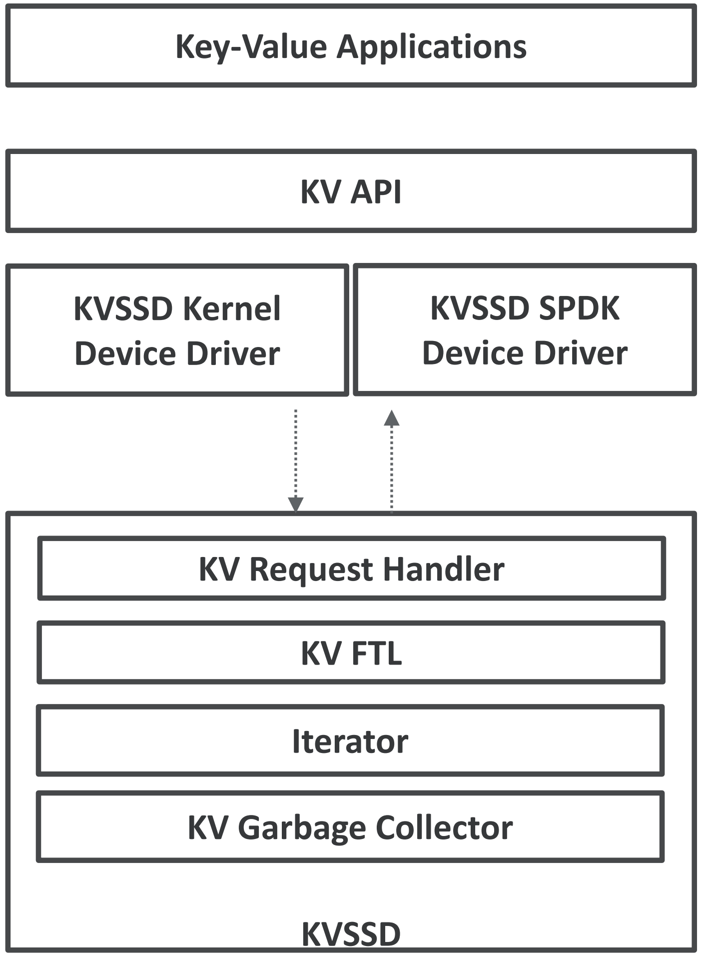
Goals

- **No dirty metadata** in a host system
  - Device provides indexing: large keys and key groups
- Saturate KV-SSD with minimal CPU resources (**one CPU core**)

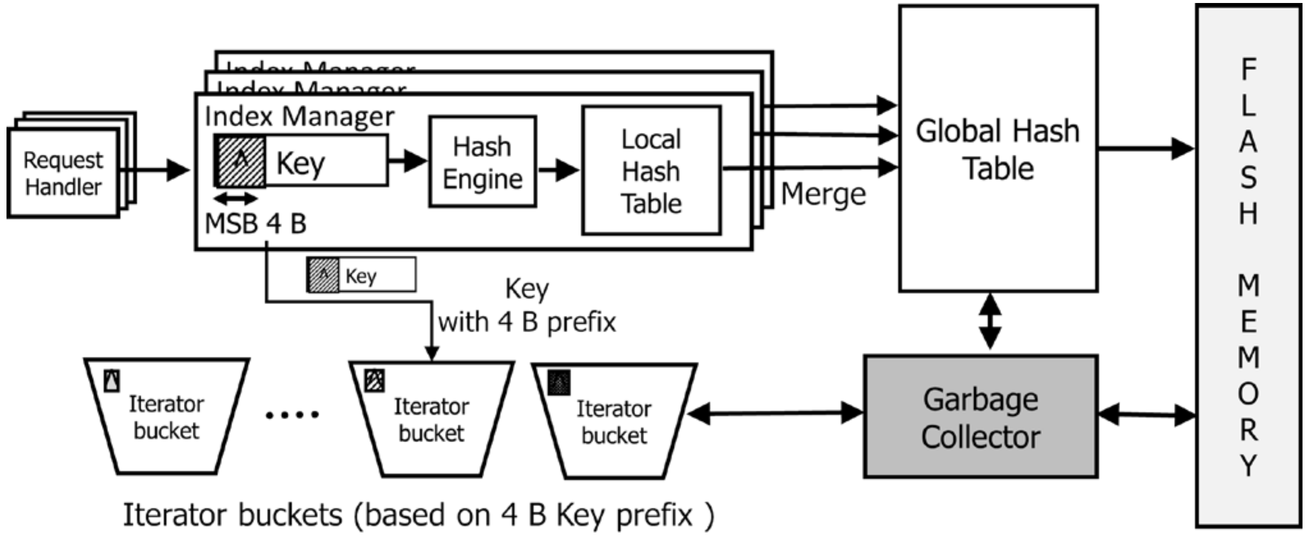
Y. Jin, H. Tseng, Y. Papakonstantinou and S. Swanson, "KAML: A Flexible, High-Performance Key-Value SSD," *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Austin, TX, 2017, pp. 373-384.

Key-Value SSD

- Our KV-SSD prototype is implemented as SSD firmware that runs on the existing the block NVMe SSD hardware
  - Block SSDs can be switched to KVSSDs
- Main components
  - Key-Value Request Handler
  - Hash-based FTL
  - Iterators
  - Garbage Collector for key-value pairs
- KV API and driver
  - API and driver for key-value SSDs are available in github
  - SNIA Standard : https://www.snia.org/tech_activities/standards/curr_standards/kvsapi



Supporting Variable-length Key and Key Groups in KV-SSD

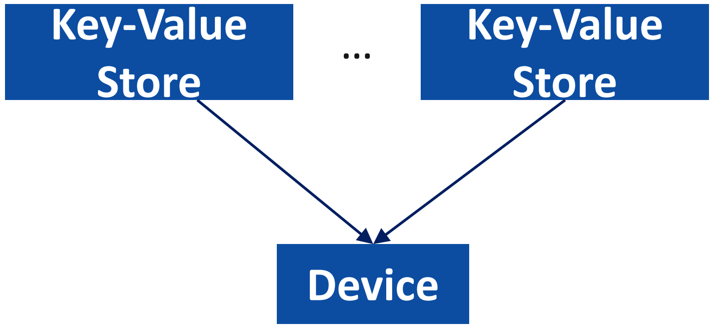


- Use of Hash-based data structure
  - Limit the memory per key-value pair in SSDs
- Global and Local Hash tables
  - Reduce lock contention between Index Managers
  - Advanced features (e.g. transactions)

- Key Groups
  - First 4B of the key is used as an index of a group
  - Stored to the write buffer and later flushed to an iterator bucket identified by the prefix

Evaluation

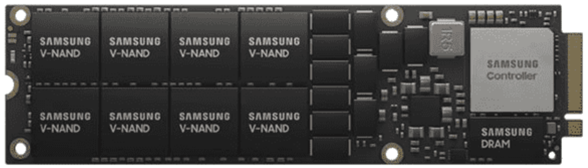
- We compare the performance and resource utilization of KVSSDs against the thee state-of-the-art key-value stores using up to 18 NVMe devices
  - Rocksdb , Rocksdb + SPDK , AeroSpike
- Key-Value Store Configuration with multiple SSDs
  - Assigned multiple instances per device to saturate the bandwidth

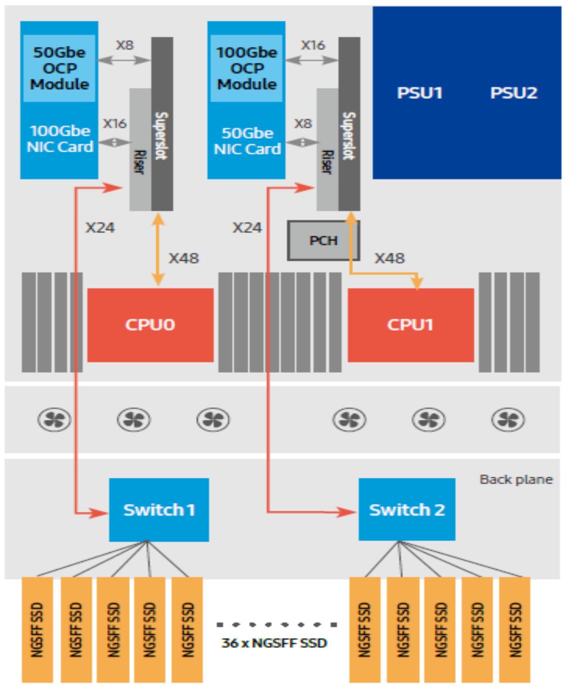


- Key-Value Benchmark, KVSB
  - Launch all instances of key-value stores at once
  - NUMA-aware CPU and memory assignment (core and memory pinning when needed)
  - Support sequential / uniform / zipfian distributions and YCSB A,B,C and D workloads

Evaluation Environment

- Server: Custom-designed storage server for PCIe SSDs (Mission Peak System)
  - 2 Xeon E5 2.1GHz CPUs (48 cores with hyper-threading per CPU)
  - 768 GB DRAM
- 18 PCIe SSDs are attached to one CPU
  - Samsung PM 983 PCIe SSD
  - Same hardware is used for both host key-value stores and KV-SSDs



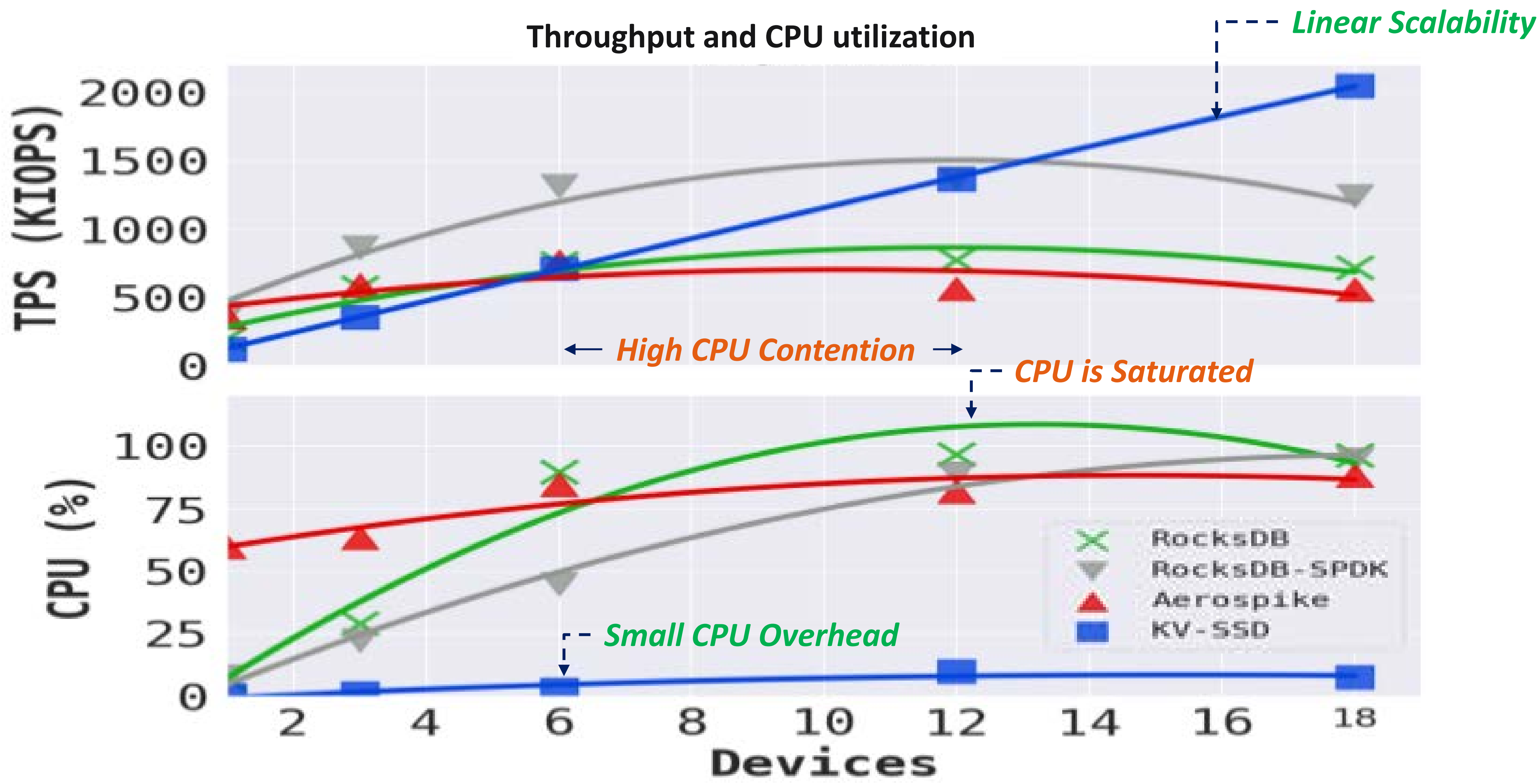


Mission Peak System*

(Support up to 18 PCIe SSDs Per CPU)

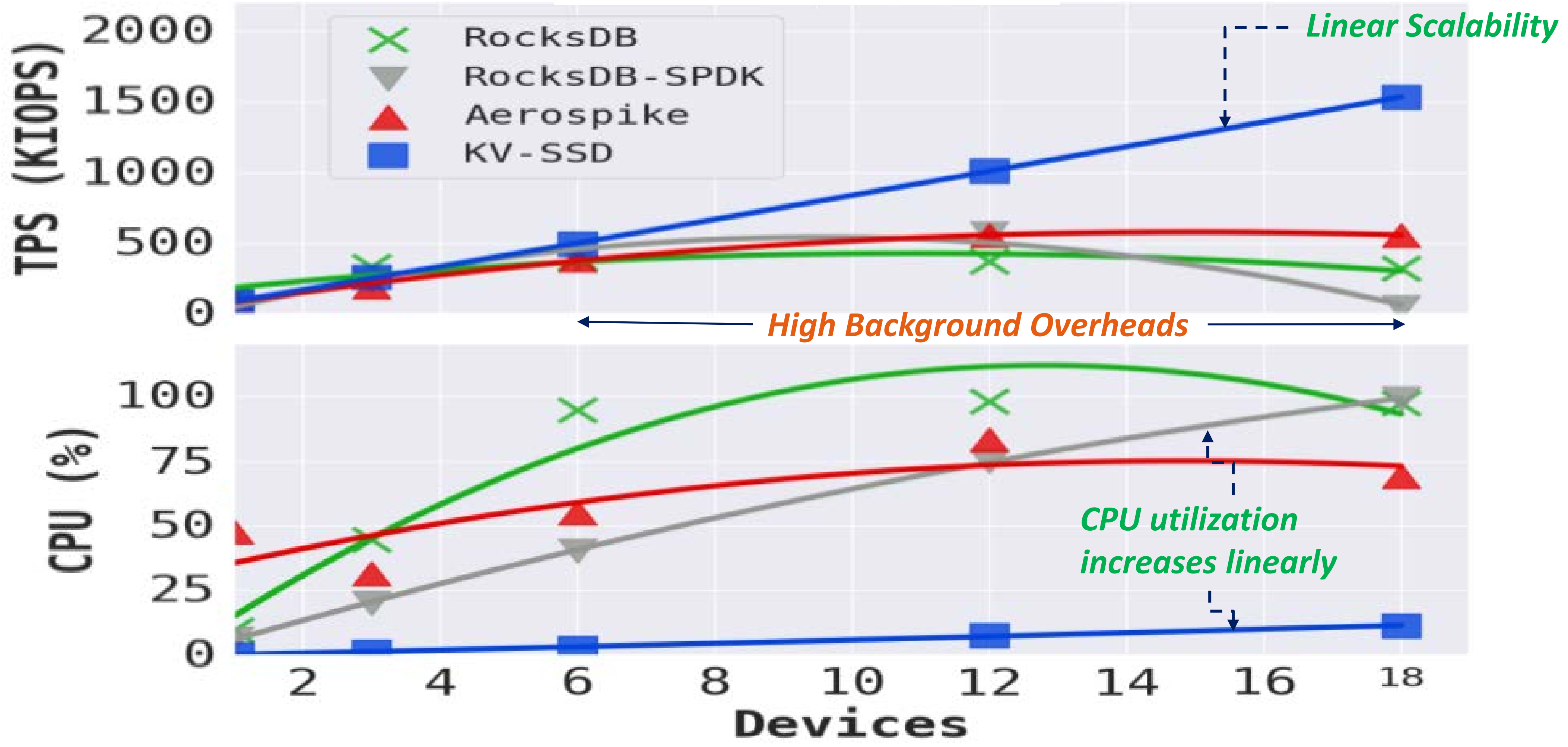
* https://www.samsung.com/semiconductor/global.semi.static/Whitepaper_Mission_Peak_Reference_Server_System_for_NGSFF_SSD_1809.pdf

Sequential Workloads (Small Background Overheads)

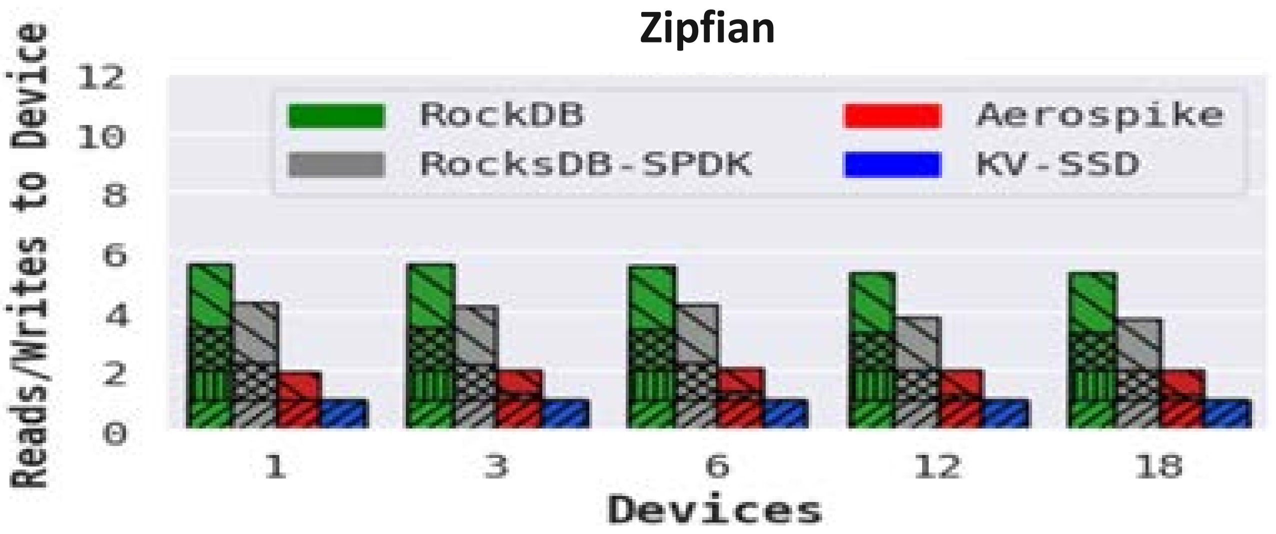


Zipfian Workloads (Large Background Overheads)

Throughput and CPU utilization

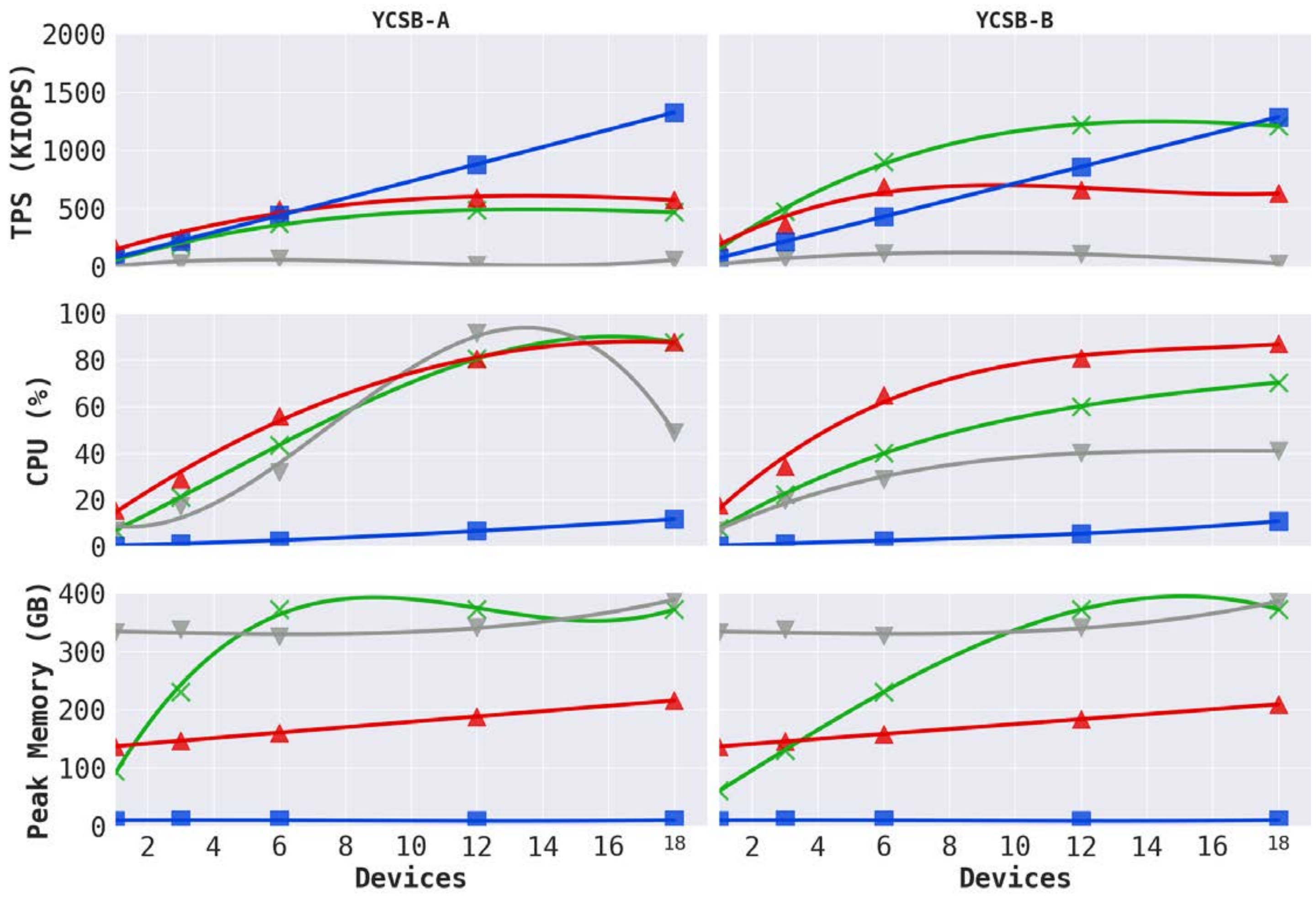


Read/Write Amplification



- Background operations write 5 times more data
  - Write-ahead Log
  - Compaction
- Read/Write amplification will make devices busy but the throughput will become lower

YCSB Workloads (Mixed Reads and Writes)

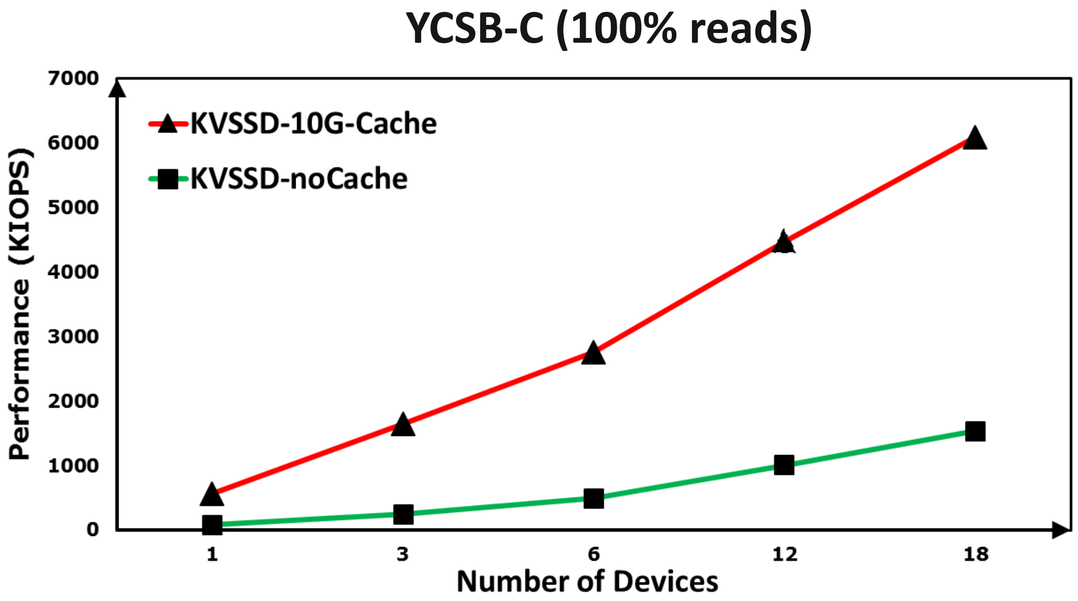


50% read – 50% write
Zipfian Distribution

95% read – 5% write
Zipfian Distribution

- KVSSD *without cache* scales linearly providing comparable performance
- YCSB-A suffers from high background processing overheads
- Host key-value stores' read cache shows high memory usage

Effects of Caching



- Read cache can be easily added to KVSSD ecosystems for better performance
- With 10GB of LRU cache per device (180 GB total),
  - 4 times better read performance per device
  - 50% lower memory footprint than other key-value stores
- Where do the benefits come from?
  - No Lock contention
  - Finer-grained caching (key-value vs. page)

Conclusion and Future Work

- **Summary**
  - High resource demands of host key-value stores make it difficult to utilize fast SSDs
  - The performance of KV-SSDs scales linearly while requiring significantly lower host-system resources and outperforming conventional host-side key-value stores

- **Standardization**
  - Key Value Storage API Specification (SNIA): https://www.snia.org/tech_activities/standards/curr_standards/kvsapi
  - Key-Value Device Commands: submitted for a review to NVMe standard committee

- **KV APIs and Drivers**
  - Available at **https://github.com/OpenMPDK/KVSSD**
  - Key-Value SSD will become commercially available soon

Thank you