



Flash Memory Summit

Challenges In Designing KV SSD

Vishwas Saxena

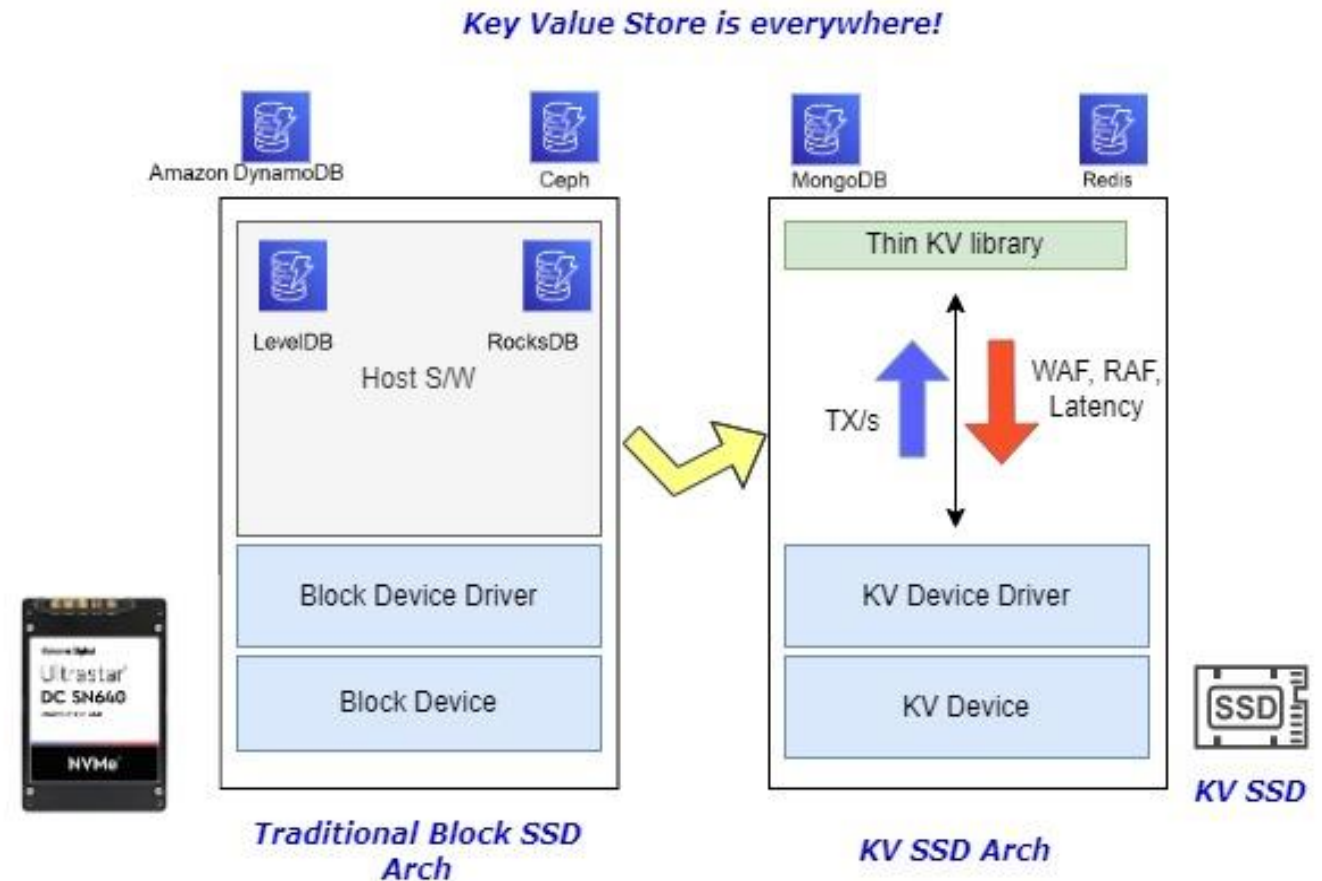
Senior Technologist, Firmware Engineering, Western Digital

KV Stores Are Ubiquitous In Unstructured Data Storage



Flash Memory Summit

- Web Indexing, Caching – Twitter, Facebook, LinkedIn
- Media – Images, Videos
- Document Stores
- Blockchain Data



How Are They Useful ?



Flash Memory Summit

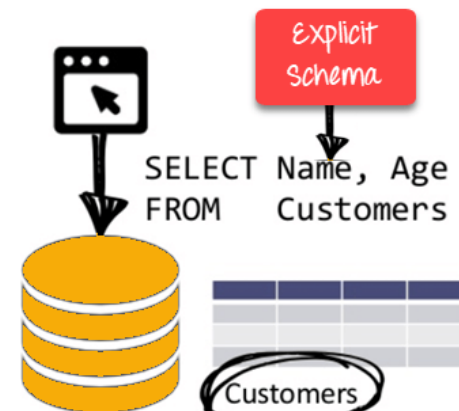
Nature of Big Data

- Big data is all unstructured and access is heavy on writes and light on reads
- Big Data storage needs to scale smoothly as volume increases
- KV map provides an easy representation of data providing $O(1)$ write operations

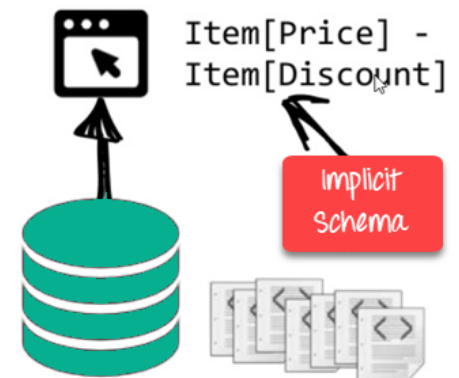
KV Apps and Comparison with Relational DB



RDBMS:



NoSQL DB:



Log Structured Hash

- Supports iterate
- Requires less changes in FTL based L2P architecture
- Require more space for Hashed Mapping Tables => needs more DRAM

Log Structured Merge

- Requires HW Accelerator for merge sort
- Does not cover operation to iterate over a list of KV pairs
- Needs building Bloom Filters



Challenges In Designing KV SSD

Limited Compute and DRAM inside SSD

DRAM

- SSDs usually have DRAM as much as 0.1% of NAND for L2P Cache indexing
- DRAM scales slower than NAND and costly than NAND, can't add more

CPU

- SSD have low CPU Power
- Optimized for low power (5-15W active power consumption)

Impact Of Variable Size KV Pairs



Flash Memory Summit

- L2P Tables in KV SSD require 12 Byte Entries instead of 4 Byte entries in Block SSDs
- L2P Tables in SSD can be greatly optimized by Fixed size KV Pairs. L2P Table size affects both NAND L2P Tables and DRAM caching
- Small Size (32B, 1KB) KV Pairs greatly increases the L2P Table Size in NAND by 12 times
- Large Size (32B, 2MB) KV Pairs greatly reduces the L2P Table size in NAND by 170 times

SSD L2P table (1B per KB)								
4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry	4KB L2P Entry
4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add	4B Phy Add

L2P Table Size

L2P table size get increased by 12times

L2P Table Size Increase

4B, 1KB KV Pair

KV L2P Entry (4B, 1 KB KV Pair)
6B Hashed Key (4B Hash + 2B Iterate Prefix)
4B Phy Add
2B Prefix for Iterate

KV L2P Entry (4B, 2MB KV Pair)
4B Hashed Key, 2B Len
4B Phy Add
2B Prefix for Iterate

4B, 2MB KV Pair

L2P Table Size Decreases

L2P Table Size

L2P Table size gets reduced by 170 times (2048/12)

KV SSD L2P Table							
KV L2P Entry	KV L2P Entry	KV L2P Entry	KV L2P Entry	KV L2P Entry	KV L2P Entry	KV L2P Entry	KV L2P Entry
12B Hash+Adr	12B Hash+Adr	12B Hash+Adr	12B Hash+Adr	12B Hash+Adr	12B Hash+Adr	12B Hash+Adr	12B Hash+Adr

Need For Cuckoo Hashing?



- Hash collisions are costly and impacts KV SSD architecture
- Need separate mapping tables for collisions in cache and NAND.
- Cuckoo Hashing is a simple scheme for
 - Resolving collisions that allows for constant-time worst-case lookup and deletion operations
 - Amortized constant-time insertion operations
 - Maintains two hash tables T_1 and T_2 with two independent hash functions h_1 and h_2 , respectively, and a key is stored in exactly one of two locations, with one possible location in each hash table

- Maintain two tables, each of which has m elements.
- We choose two hash functions h_1 and h_2 from \mathcal{U} to $[m]$.
- Every element $x \in \mathcal{U}$ will either be at position $h_1(x)$ in the first table or $h_2(x)$ in the second.

	97
32	
	26
84	
59	41
93	23
58	
	53
T_1	T_2

- To insert an element x , start by inserting it into table 1.
- If $h_1(x)$ is empty, place x there.
- Otherwise, place x there, evict the old element y , and try placing y into table 2.
- Repeat this process, bouncing between tables, until all elements stabilize.

	97
53	
	26
6	32
75	93
10	23
58	
	84
T_1	T_2

ZNS – Nice Alternative To Improve KV Latency

- Leverage Host ecosystem - ZenFS etc
- Interlocked host/device protocol, where the host manages placement and levels of OP can approach 0%
- Deliver maximum capacity and lowers SSD Cost
- ZNS solves
 - Data Center KV Store Latency problem
 - Storage disaggregation problem
- But the same problem exists in client and consumer workloads as
 - More and more apps caches media
 - Unstructured text used by Conference Apps

- SSD CPU and DRAM are scarce resources
 - Hashing, Merge, Sort operations of KV SSD can't work with limited CPU and DRAM resources of SSD
 - Need HW Accelerators for improved hashing, merging, sorting
- Variable size KV Pairs are not optimal for KV SSD L2P Tables
- ZNS ecosystem provides a nice alternative to solve the system latency problem of KV operations
- KV operations in Client SSD workloads on PCs remain a challenging problem to solve with limited CPU and DRAM resources of SSD



Thank You

Western Digital is a registered trademark or trademark of Western Digital Corporation or its affiliates in the US and/or other countries. Amazon DynamoDB is a trademark of Amazon.com, Inc. or its affiliates in the United States and/or other countries. Ceph and the Ceph logo are registered trademarks of Red Hat Inc. in the U.S. and other countries. MongoDB and the MongoDB logo are trademarks of MongoDB, Inc. Redis is a trademark of Redis Labs Ltd. All other marks are the property of their respective owners