RHIK: Re-configurable Hash-based Indexing for KVSSD

Manoj P. Saha
Florida International University
Miami, Florida, USA
msaha002@fiu.edu

Bryan S. Kim
Syracuse University
Syracuse, New York, USA
bkim01@syr.edu

Haryadi S. Gunawi
University of Chicago
Chicago, Illinois, USA
haryadi@cs.uchicago.edu

Janki Bhimani
Florida International University
Miami, Florida, USA
jbhimani@fiu.edu

ABSTRACT

Key-Value Solid State Drive (KVSSD), a key addressable SSD technology, promises to simplify storage management for unstructured data and improve system performance with minimal host-side intervention. However, we find that the current state-of-the-art KVSSD exhibits indexing peculiarities that limit their widespread adoption. Through experiments, we observe that the performance degrades as more data are stored, and the KVSSD can only store a limited number of key-value pairs even though the amount of data stored on the device is significantly lower than its capacity. We introduce RHIK, a reconfigurable hash-bashed indexing for KVSSD, for high performance and high occupancy. We implement our proposed indexing scheme on the open-source KVSSD emulator that is validated against a real KVSSD, and demonstrate its effectiveness using real workload traces and synthetic microbenchmarks.

CCS CONCEPTS

• **Information systems → Flash memory**.

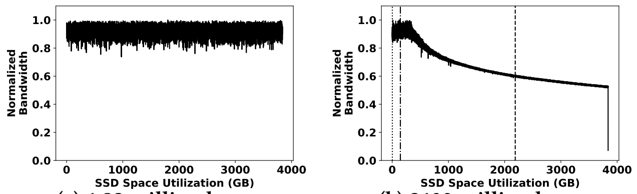
KEYWORDS

Key-Value SSD, data storage, KV indexing

ACM Reference Format:
Manoj P. Saha, Bryan S. Kim, Haryadi S. Gunawi, and Janki Bhimani. 2023. RHIK: Re-configurable Hash-based Indexing for KVSSD . In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23), June 16–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3588195.3595945

1 MOTIVATION

Performance drops as index size increases: The hash-based index in current KVSSD increases the tail latency and hampers performance as the size of the index grows [2] because the index becomes too large to fit in the SSD DRAM and optimizations from block-SSD cannot be applied directly. This behavior is visible in Fig. 1. Starting in Fig. 1a, KVSSD can sustain I/O performance at a smaller index size, but as the index grows from 2 million vs. 116 million vs. 1760 million vs. 3100 million (patterned vertical lines as shown in 1b) KVSSD's performance collapses.



(a) 1.83 million keys (b) 3100 million keys

Figure 1: Write bandwidth drops with increasing index size

Index supports only a limited number of keys: We observe that KVSSD also supports only a limited number of keys compared to its capacity. Even though Samsung KVSSD supports an unfathomably large keyspace (consisting of $2^4 + 2^5 + ... + 2^{254} + 2^{255} keys$), our experiments reveal that a 3.84TB PM983 KVSSD can store a maximum of approximately 3.1 billion KV pairs (with value lengths of 1KB or lower). Thus, we next analyze the number of keys required by most of the existing KV stores. The index in a 4TB KVSSD would need to store 34 million-2.7 billion keys for a typical Baidu Atlas KV workload [3] and 24-744 billion keys for a typical Facebook Memcached workload [1].

2 RHIK ARCHITECTURE

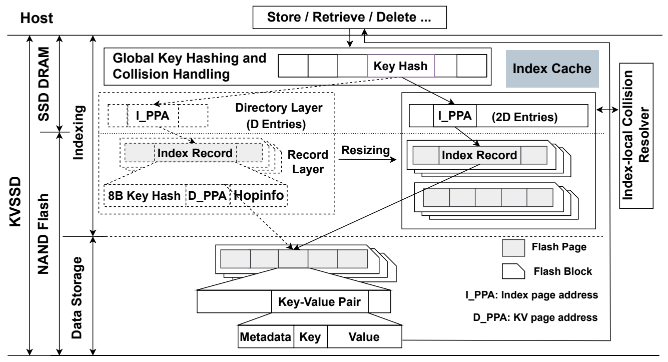


Figure 2: Re-configurable indexing

We introduce RHIK, a recofingurable hash-based indexing for KVSSD, and to the best our knowledge the first KVSSD design that guarantees maximum of one flash read for index access and supports high index occupancy. The first layer in RHIK works like a directory, containing *D* entries, and accessed from SSD DRAM (Fig. 2). At the same time, a periodically updated persistent copy of these *D* entries resides on flash. The second layer, or record layer, holds fixed-size independent hash tables that store the metadata of corresponding KV pairs and is served from flash unless available in the integrated RAM. With this 2-level design, we are able to guarantee maximum of one flash read per index access, something that is almost impossible to achieve in the current state-of-the-art multi-level hash index or LSM-tree design. In addition, RHIK has the three other important features - efficient collision management, conservative index initialization and resizing, and low-overhead membership checking.

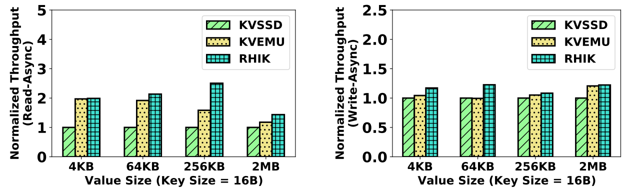
Efficient Collision Management: RHIK employs Hopscotch hashing to ensure better collision handling and higher translation page (i.e. NAND flash pages storing the index records) occupancy (Fig. 2). While RHIK's collision management scheme needs marginally more compute than existing collision handling mechanism, it reduces the number of translation pages required for storing the index.

Conservative Index Initialization and Resizing: RHIK is initialized conservatively to reduce space wastage related to provisioning a large under-utilized hash index, and resizing triggered only when total occupancy reaches a pre-defined threshold (Fig. 2).

Low-overhead Membership Checking: RHIK reuses key signatures for membership checking, instead of traditional Bloom filter-based approaches (Fig. 2). This reduces the memory footprint and management complexity of the index.

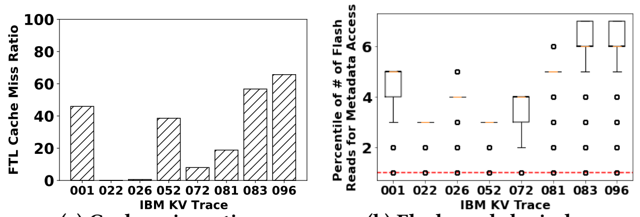
RHIK on KV Emulator We develop an advanced version of the KV Emulator by extending OpenMDK KV Emulator. The new KV Emulator imitates the fundamental hardware primitives of an SSD, such as flash blocks, pages, etc. We implement RHIK in the new KV Emulator. We implement separate indexing and data layout schemes.

3 EVALUATION



(a) Asynchronous reads (b) Asynchronous writes

Figure 3: I/O performance comparison



(a) Cache miss ratio (b) Flash reads by index

Figure 4: RHIK incurs only one flash read per I/O request

First, we evaluate RHIK 's performance using multiple sequential workloads with variable key-value sizes (Fig. 3). Since the KV Emulator is limited by host DRAM capacity, we evaluate RHIK for 100 million keys on a 64GB emulated drive. When we compare asynchronous write performance, for almost all value sizes, RHIK achieves higher throughput. For asynchronous read requests, we observe that RHIK is able to perform better with large value sizes during read. We observe that the performance trends of the normalized throughput of the OpenMPDK KV Emulator differs from KVSSD. We believe that this difference in the performance trends may be due to the IOPS model used by the OpenMPDK KV Emulator. Since, our extended KV Emulator also uses the same IOPS model, we measure RHIK 's performance improvements relative to OpenMPDK KV Emulator performance. In addition, KVSSD and RHIK both maintain un-ordered index with KV pairs written in a log-like manner on flash, and KV operations are largely dominated by key handling operations, the performance of sequential workloads are not significantly different from Uniform or Zipfian workloads in KVSSD for most cases.

As we show in the motivation, the KVSSD performance drops when multi-level hash index size increases due to multiple flash page accesses needed to read the portion of the index that doesn't fit within the SSD DRAM. Fig. 4 shows that the number of flash page accesses using RHIK is significantly less compared to that of the multi-level hash index. Particularly, here we limit the SSD DRAM cache budget to 10MB (i.e. for a 10GB SSD) for both and we replay the real-world IBM Cloud Object Store KV traces. Out of the eight clusters, four clusters (i.e. 022, 026, 052, and 072) index fits within the SSD cache, so no difference between RHIK and KVSSD is observed. However, for the rest of the clusters that have an index size of more than 10MB, RHIK incurs only one flash access but multi-level hash within KVSSD may incur 3 to 7 accesses. Hence, the performance using RHIK will not drop upon increasing index size.

4 CONCLUSION

KVSSDs established the performance benefits of removing the I/O stack redundancies for KV workloads and finally became a commodity product in 2018, but it is considerably more expensive than block SSDs. However, as we show unoptimized index management can make the KVSSD underutilized and lower its performance to be worst than the block SSDs. In this work, we show novel index management along with many other features needed to achieve sustained performance from KVSSDs even when the index size increases.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation (NSF) Awards CNS-2008324 and CNS-2122987, and KV-SSD equipment grant and funded research collaboration with Samsung MSL GRO(Global Research Outreach).

REFERENCES

[1] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-Scale Key-Value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[2] Junsu Im, Jinwook Bae, Chanwoo Chung, Arvind, and Sungjin Lee. 2020. PinK: High-speed In-storage Key-value Store with Bounded Tails. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 173–187. https://www.usenix.org/conference/atc20/presentation/im

[3] C. Lai, S. Jiang, L. Yang, S. Lin, G. Sun, Z. Hou, C. Cui, and J. Cong. 2015. Atlas: Baidu's key-value storage system for cloud data. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. 1–14.