2018 7th IEEE Non-Volatile Memory Systems and Applications Symposium

NVLH: Crash-consistent Linear Hashing for Non-Volatile Memory

Hu Wan*, Fuyang Li*, Zimeng Zhou*, Kaisheng Zeng†, Jianhua Li‡, Chun Jason Xue*
*Department of Computer Science, City University of Hong Kong, Hong Kong, China
†School of Information and Communication, National University of Defense Technology, Wuhan, China
‡School of Computer Science and Information, Hefei University of Technology, Hefei, China

I. Introduction

Linear hashing [1] is a dynamic hashing scheme. It provides stable performance, high storage utilization, and supports both expansions and contractions. Linear hashing has been widely used in disk-oriented database systems, including the BerkeleyDB and PostgreSQL. These databases rely on recovery from disk. For workloads with large datasets, reading data from disk to rebuild index state after a restart can take up several hours [2]. Emerging non-volatile memory (NVM) technologies, such as memristors [3] and 3D XPoint [4] open new opportunities to persist linear hashing index, thus drastically reducing the warm-up time. These storage technologies enable data to be stored on NVM persistently, and therefore a database can access the indexes directly on NVM after a restart or crash without reindexing the database first. A database can also directly access persistent data via load/store instructions, without performing expensive block I/O as well as format transformation between memory and disk data structures. However, exploiting non-volatility to support persistence is not as simple as just keeping data on NVM. It also introduces non-trival programming challenges, such as crash consistency mechanisms. Unfortunately, there is no existing work that investigates comprehensive designs and optimizations for linear hashing to make full use of NVM. This work revisits linear hashing and presents the design of NVLH which is a persistent and crash-consistent linear hashing for NVM.

II. Linear Hashing Revisited

Linear hashing is a dynamic hashing scheme for gradually expanding (and contracting) the storage area of a hash file. Under insertions, the file gracefully expands through the splitting of one bucket into two buckets. Linear hashing accomplishes this by using a pair of hash functions $h_i(key)$ and $h_{i+1}(key)$ (i = 0, 1, 2...) to calculate addresses. Each function's range is twice that of its predecessor. Splitting occurs one by one from the first bucket to the last bucket. A special pointer p is used to keep track of the next bucket to be split. When all the original buckets have been split, the pointer p is reset to zero, and the splitting process starts over again.

Fig. 1 illustrates the process of inserting and splitting of linear hashing. Initially, linear hashing calls $h_0(key)$ to distribute eight records into four buckets. After that, the insert of 21 into bucket 1 causes a collision. An overflow bucket is chained to the primary bucket to contain the inserted value. The first bucket indicated by pointer p is split, and its records are redistributed using $h_1(key)$ into two buckets, i.e., original bucket 0 and new bucket 4. If $h_0(key)$ maps a value to a bucket which is smaller than p, $h_1(key)$ must be used to insert the new entry, e.g., the insert of 20 in Fig. 1(c). With the insertion of keys 23, 27 and 25, buckets 1, 2 and 3 are split in turn. At this point, all the four buckets are split. p is reset to zero, and a new round starts. In the second round, $h_0(key)$ will be deprecated, and a new hash function $h_2(key)$ will be used.

The linear hashing operations, such as creating an index, inserting, deleting, bucket splitting, and allocating overflow buckets, don't guarantee hash index consistency after a crash even if they are stored on NVM. Therefore, persistence and consistency support should be provided to provide robustness.

III. NVLH

NVLH builds on Direct Access (DAX) feature of operating systems which allows applications to access NVM directly via memory mapping. This also enables fast warm-up of index states after a restart. NVLH guarantees crash-consistency of the index with the help of durable transaction.

1) Persistence support: NVLH assumes a single-level store model where NVM is attached directly to the memory bus as part of the main memory in addition to DRAM. The operating system provides DAX support for mapping NVM in application address space via the memory mapping interface. Applications can thereafter access NVM with simple `LOAD`/`STORE` instructions. More specifically, NVLH prefers device DAX to filesystem DAX since it gives the user full control and allows memory ranges to be allocated and mapped even without intervening file system. Device DAX supports userspace flushing of NVM stores to guarantee durability and ordering. This is done through a cache line flush instruction (e.g., `CLFLUSH`), or a combination of weakly-ordered cache line flush instructions (e.g., `CLFLUSHOPT` and `CLWB` on Intel processors) and memory fences.

2) Consistency support: NVLH introduces crash-consistent durable transaction for each of these operations that need to update data on NVM. A durable transaction makes a group of persistent memory updates appear as one atomic unit with respect to a system crash. NVLH implements transactions with undo logging which stores the original values to the log before a transaction modifies them. As splitting involves multiple actions, it is possible that the system crashes between

2575-257X/18/$31.00 ©2018 IEEE
DOI 10.1109/NVMSA.2018.00031

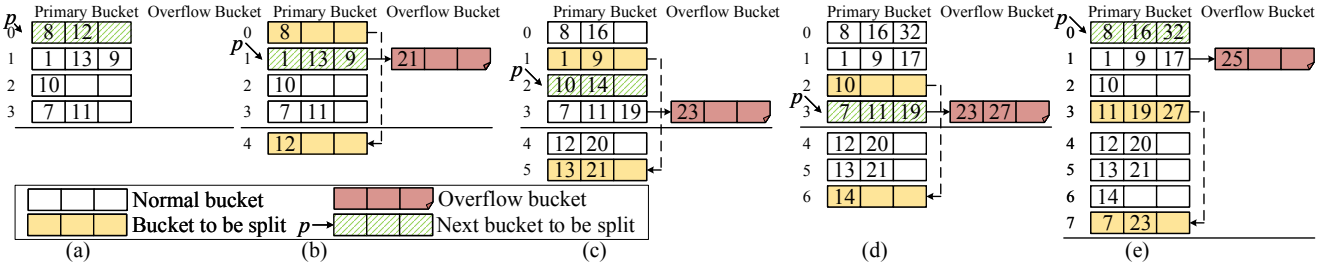

Fig. 1: **Linear hashing example.** A linear hashing uses hash functions $h_0(key) = key\%4$ and $h_1(key) = key\%8$ to caculate address, and splits bucket when an overflow occurring. (a) A linear hashing file with 8 keys. (b) Insert of 21 causes a collision and split of bucket 0. (c) Keys 20, 14, 19, and 16 inserted; insert of 23 causes split of bucket 1. (d) Keys 17 and 32 inserted; insert of 27 causes split of bucket 2; (e) Insert of 25 causes split of bucket 3.

moving records from old bucket to new bucket. In such a case, the old and new buckets will be marked with in-process flags which indicates that split is in progress for recovery. Durable transaction works as follows: `tx_begin` opens a new transaction. The data to be modified is copied into log area by `tx_addlog`. After that, data can be modified as needed. `tx_commit` commits the transaction and cleans up log area. At last, `tx_end` ends the transaction.

IV. Preliminary Evaluation

We compare NVLH against NVDH which is a double hashing scheme for NVM which periodically rehashes records into a larger table [5]. We perform 100,000 key-value pairs insertion, deletion or query operations each time. Since non-volatile memory is not yet widely available, we simulated NVM by memory mapping via a /dev/dax0.0 character device created by the `ndctl` utility into NVLH's address space.

Fig. 2 shows the performance of the experimental hashing schemes on NVM as we vary the size of values. The benefits of NVLH are more prominent in the insertion and deletion, where NVLH's throughput is 1.31× and 1.34× higher than that of NVDH on average. NVLH performs comparatively well for query operations. Response time is also as important as throughput. For this metric, the tail of the response time distribution is considered. While the results for NVLH are similar to NVDH in the 99th and 99.9th percentile latency (not shown), NVLH provides a significant responsiveness advantage versus NVDH regarding the maximum latency. As shown in Fig. 3, the maximum latency of NVLH is one to two orders of magnitude shorter than that of NVDH for insertion and deletion. Fig. 4 shows the time breakdown for insertion, deletion, and query operations. There are no consistency overheads for query operations since they never modify data on NVM. However, significant performance overheads of adding consistency can be found for insertion and deletion. For insertion, the consistency overheads are about 62% and 53% for NVDH and NVLH, respectively. For deletion, the overheads are 40% for both NVDH and NVLH. This is mainly because many insertions/deletions cause long and complicated bucket split/merge processes, leading to high transaction logging and committing overheads. In future, we consider reducing the overheads induced by these kinds of operations.

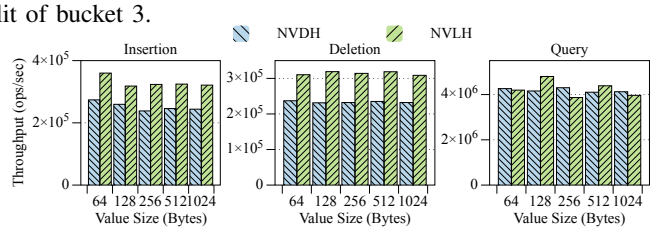

Fig. 2: Performance with varying size of value.

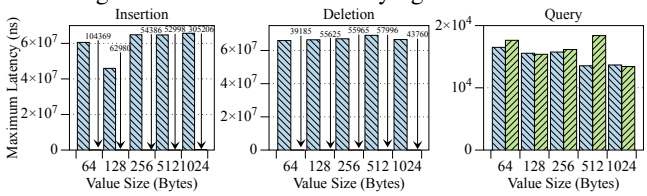

Fig. 3: Maximum latency with varying size of value.

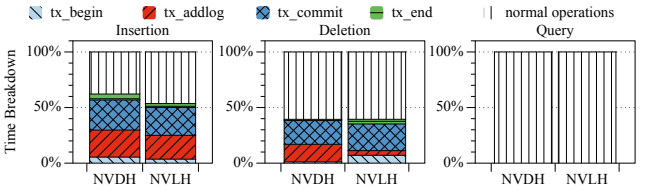

Fig. 4: Time breakdown for insertion, deletion, and query.

V. Conclusion

This paper proposes NVLH, a new design that adds persistence and consistency support to linear hashing by combining it with NVM. Preliminary evaluation shows that NVLH improves insertion, deletion performance compared to the state-of-the-art persistent double hashing scheme.

References

[1] W. Litwin, "Linear hashing: a new tool for file and table addressing." in *VLDB*, vol. 80, 1980, pp. 1–3.

[2] A. Goel, B. Chopra, C. Gerea, D. Mátáni, J. Metzler, F. Ul Haq, and J. Wiener, "Fast database restarts at facebook," in *SIGMOD*. ACM, 2014, pp. 541–549.

[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[4] Intel, "Reimagining the data center memory and storage hierarchy," https://newsroom.intel.com/editorials/re-architecting-data-center-memory-storage-hierarchy, May 2018.

[5] Intel. PMDK: Persistent memory development kit. [Online]. Available: https://github.com/pmem/pmdk