Why is it more challenging to build a hashtable for NVM than for traditional RAM?

Answer:

In a word, NVM does not perform al well as RAM in writes operation, and we need to keep the consistency between memory and cache(which need many writes), at the meantime, resizing a hashtable is often needed and will need many write operations. So, build a hashtable for NVM is very high-overhead.

1. High Overhead for Consistency Guarantee: The hashtable in persistent memory should avoid any data inconsistency (i.e., data loss or partial updates) when system failures occur, which is very high-overhead.

2. Performance Degradation for Reducing Writes: Firstly, Memory writes in NVM consume the limited endurance and cause higher latency and energy than reads. Secondly, more writes in persistent memory also cause more cache line flushes and memory fences as well as possible logging, which will decrease the system performance.

3. Cost Inefficiency for Resizing Hash Table: Resizing is essential for a hash table to increase the size, but it will result in a large number of NVM writes with cache line flushes and memory fences in persistent memory.

What did you find to be the most interesting aspect of Level Hashing?

Answer:

1.To deal with more hash collisions, they enable each key to have two hash locations via using two different hash functions, does it will increase the overhead of hashtable update operation and influence performance?

2. The bottom level is not addressable and used to provide standby positions for the top level to store conflicting key-value items.

3. in-place resizing:

1) Firstly, we allocate the memory space with 2N buckets as the new top level and put it on the top of the old hash table.

2) Rehashes the items in the IL(third level) into the top-two levels.

3) Reclaim the memory space of the IL.