A STINT WITH HASH TABLES IN STRATA

{ SE KWON LEE, MANU VISWANADHAN, HOCHAN LEE, KARTIK SATHYANARAYANAN }
THE UNIVERSITY OF TEXAS AT AUSTIN

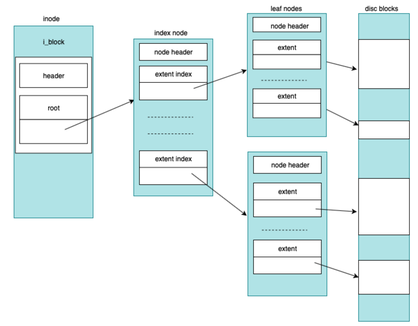
ABSTRACT

Due to the block-oriented nature of most storage devices, file systems have relied on block-oriented indexing structures, such as extent trees. With the arrival of Non-Volatile Memory (which is byte-addressable), it is now possible to use finer-grained indexing structures, such as hash tables.
We wish to replace the extent tree implementation in the NVM portion of the Strata File System with a hash table to index physical blocks

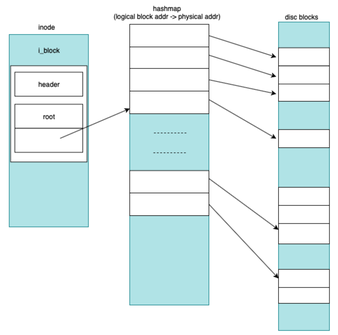
MOTIVATION

Tree-based indexing structures like extent trees, have a $O(logN)$ overhead for physical block lookup and also for modifying the structure of the tree in case of fragmentation (Here, N is the size of the data structure). Hash tables on the other hand are flat data structures that have $O(1)$ overhead on insert and delete operations.

EXTENT TREE V/S HASH TABLE



Extent tree indexing



Hashmap indexing

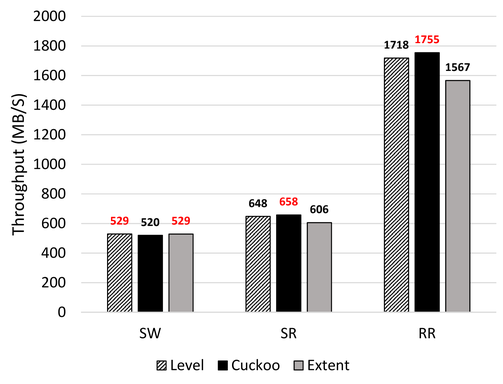
STRATA

- Multi-layered storage file system that allows for managing NVM, SSD, and HDD in unison
- Per-inode extent trees on each storage layer to index file data blocks

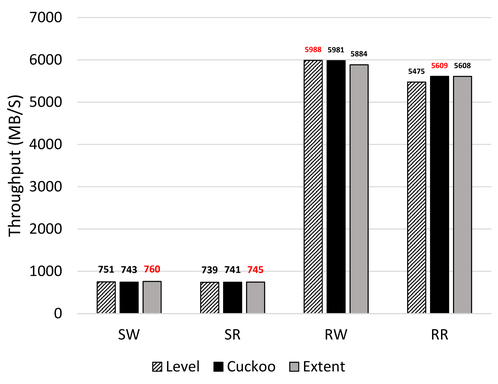
DESIGN

- Hash table maps logical block number to NVM physical block number and is stored in the shared NVM area
- Inode stores physical block number of hash table's root which is assigned during digesting inode
- Allocate 4KB blocks for files instead of contiguous chunks and update accordingly in the hash table
- Synchronize updates to hash table in the shared NVM area in a non-lazy manner

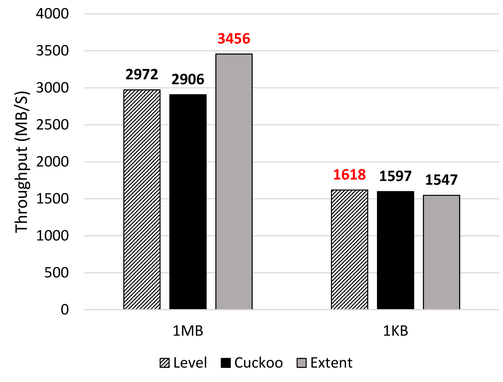
RESULTS



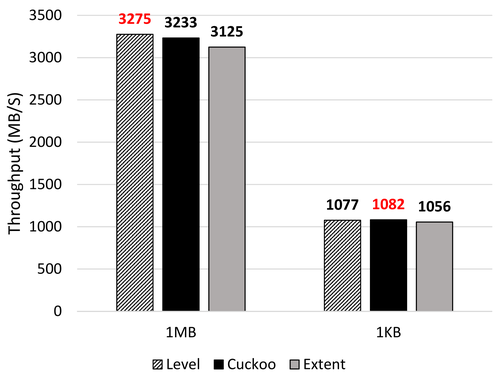
Microbechmark 1KB IO



Microbechmark 1MB IO



Webserver



Fileserver

CONCLUSION

- Hash tables better for fine IO granularity workloads due to faster search times
- Hash tables better for write and delete intensive workloads due to fragmentation immunity
- Extent trees better for sequential operations due to inherent structure
- Extent trees better for read intensive workloads with coarse granularity due to fewer number of required searches

FUTURE WORK

- Efficiently migrate NVM data to underlying block based storage while using hash table.
- Applying other recently proposed NVM-optimized indexing structures such as B+tree and Radix Tree.