

HF-Tree: A New Update Efficient Index for Flash-based Database

Da Zhou, Zhichao Liang and Xiaofeng Meng

2009-10-16



Outline

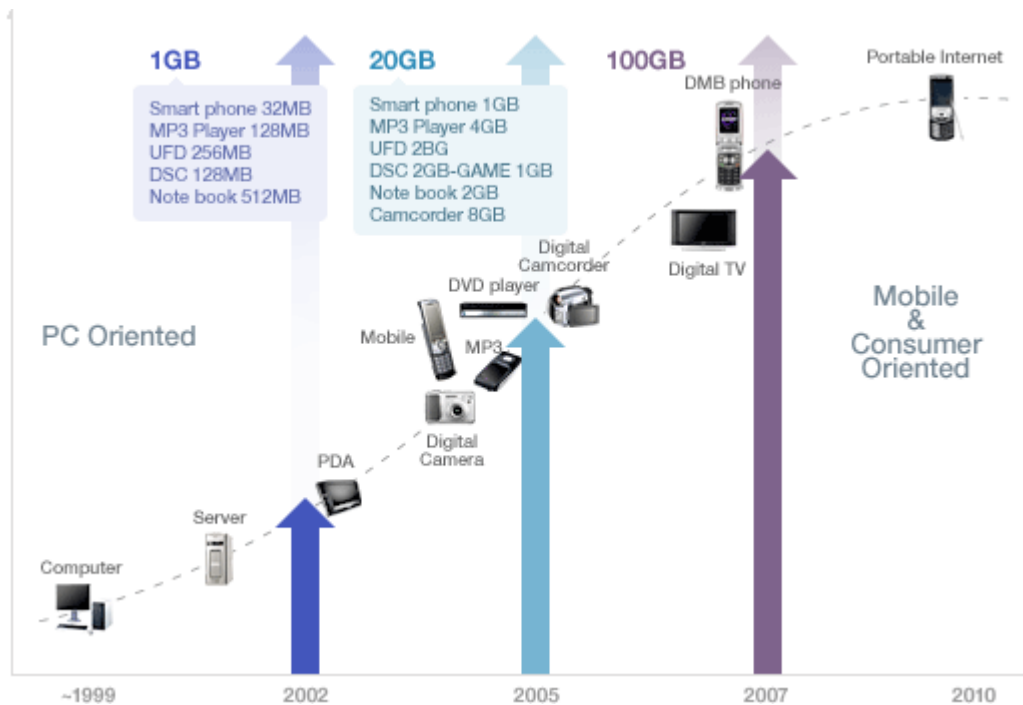
- Background
- Motivation
- Related works
- HF-Tree
- Experiments
- Conclusion



Flash disk is a popular storage medium for digital device



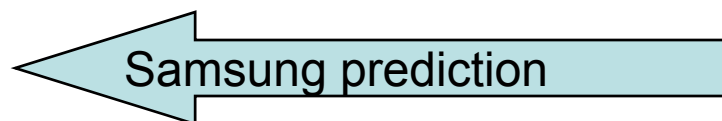
Flash disk develops at a high speed





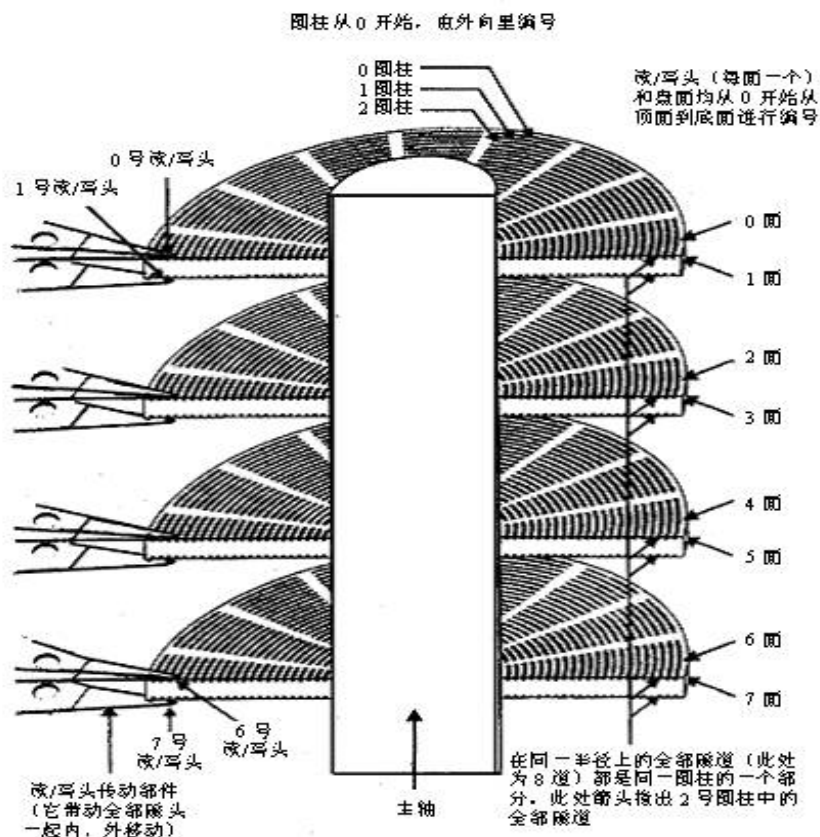
FLASH Storage

- 1995 16 Mb NAND flash chips
2005 16 Gb NAND flash
Doubled each year since 1995
- Market driven by Phones, Cameras, iPod, ...
Low entry-cost,
~\$30/chip → ~\$3/chip
- 2012 1 Tb NAND flash
== 128 GB chip
== 1TB or 2TB “disk”
for ~\$400
or 128GB disk for \$40
or 32GB disk for \$5



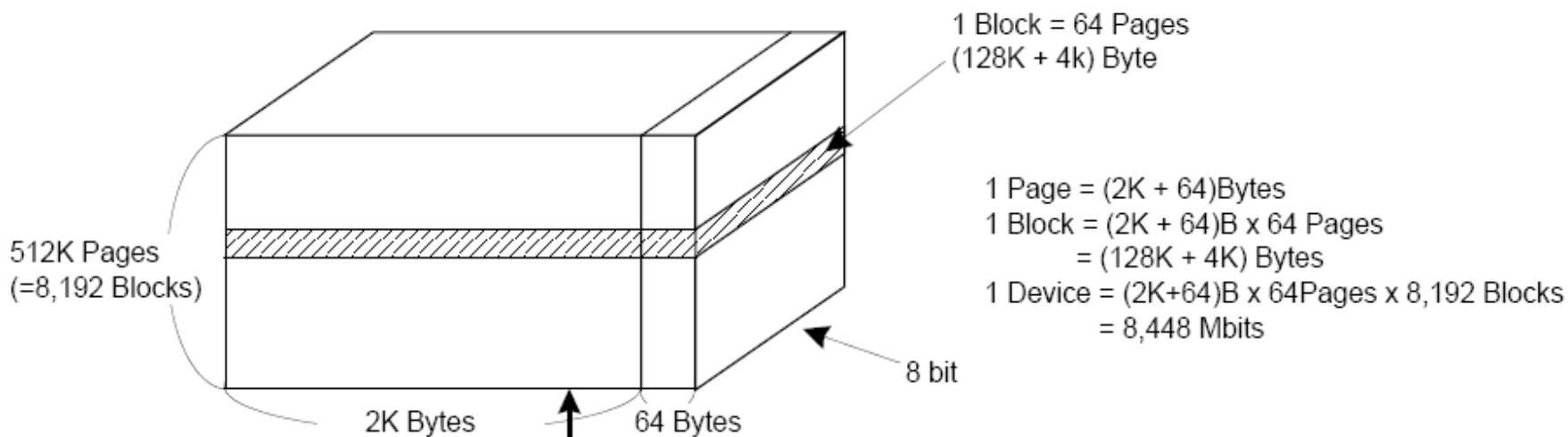


Hard Disk Hardware Architecture





Flash Memory Hardware Architecture





Outline

- Background
- **Motivation**
- Related works
- HF-Tree
- Experiments
- Conclusion



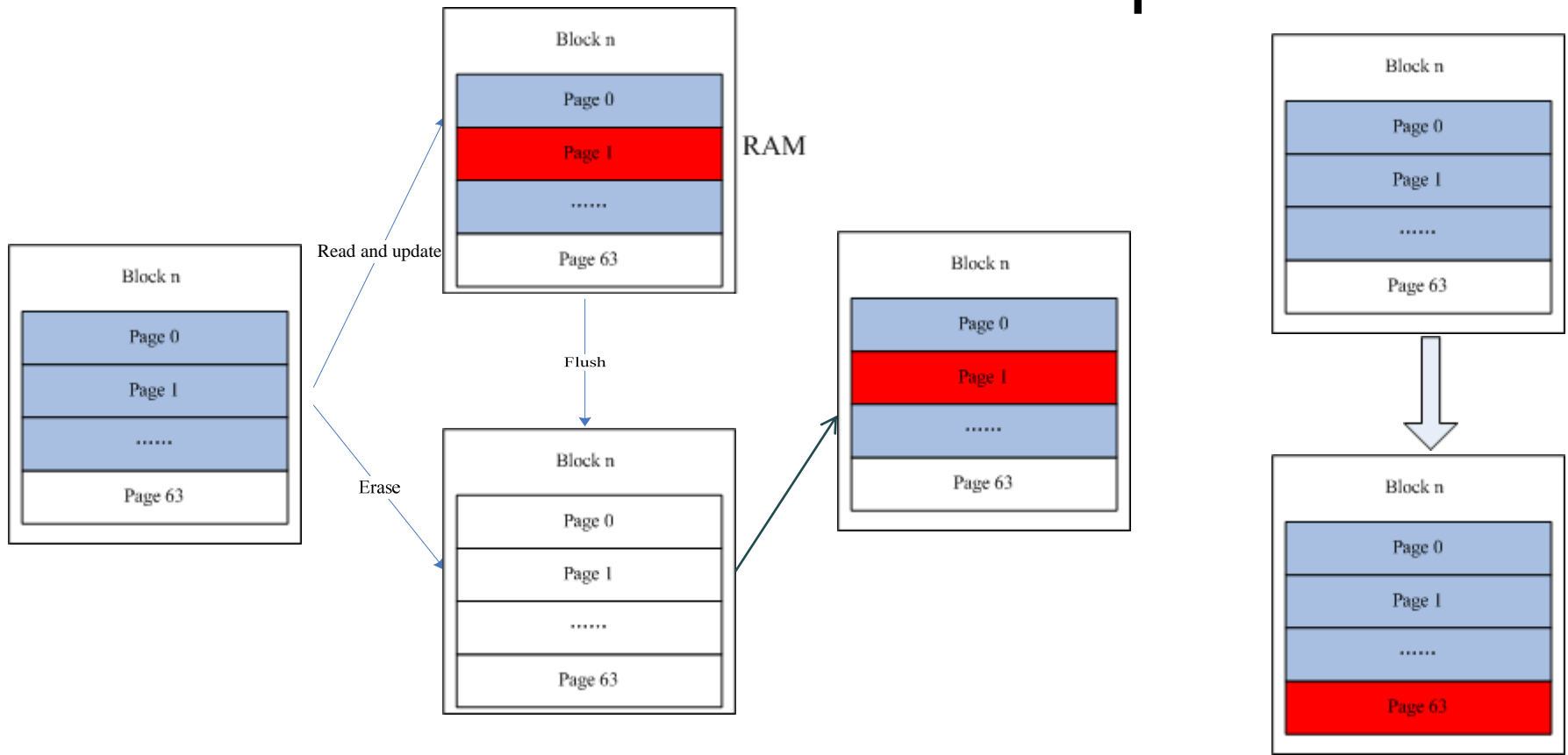
Characteristics difference between flash memory and magnetic disk

1. Asymmetric speed of read/write/erase
2. Read/Write/**Erase**
3. **Erase** before rewrite
4. Limited **erase** times
5. Read/write page, **erase** block, 1 block=64 pages

1. The same speed of read/write
2. Read/Write
3. Rewrite in-place
4. No limited write times
5. Read/write byte

Note: one flash page contains 2048 bytes

Erase → Out-of-Place update

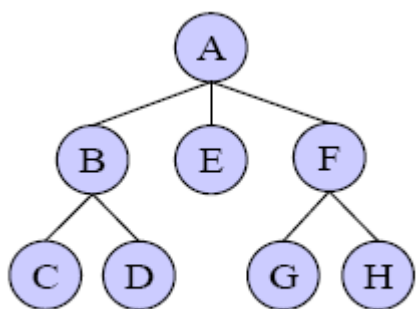


In-Place Update

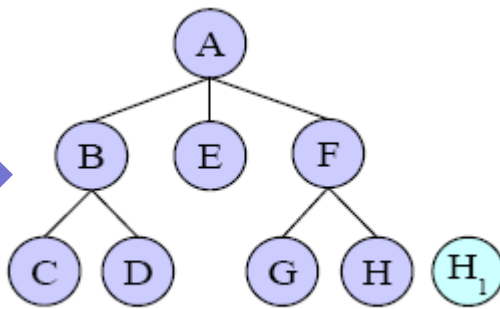
Out-of-Place Update



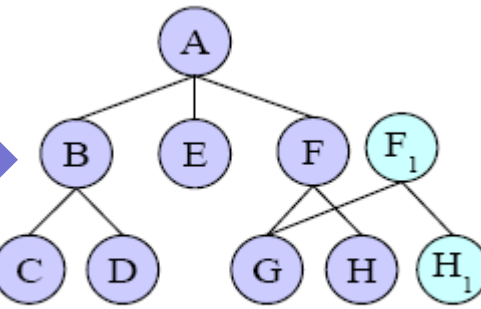
Problem of out-of-place update



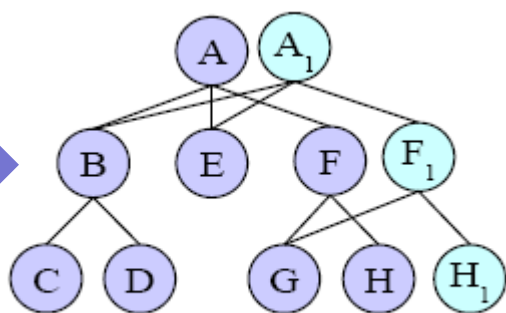
(1)



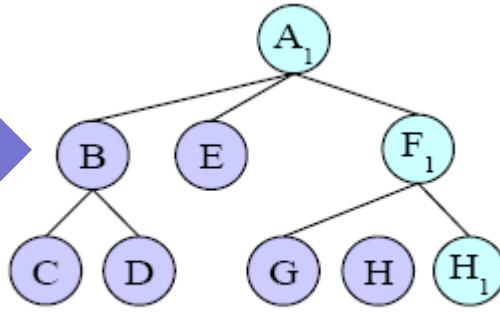
(2)



(3)



(4)



(5)

Low update performance

Large quantity of Garbage

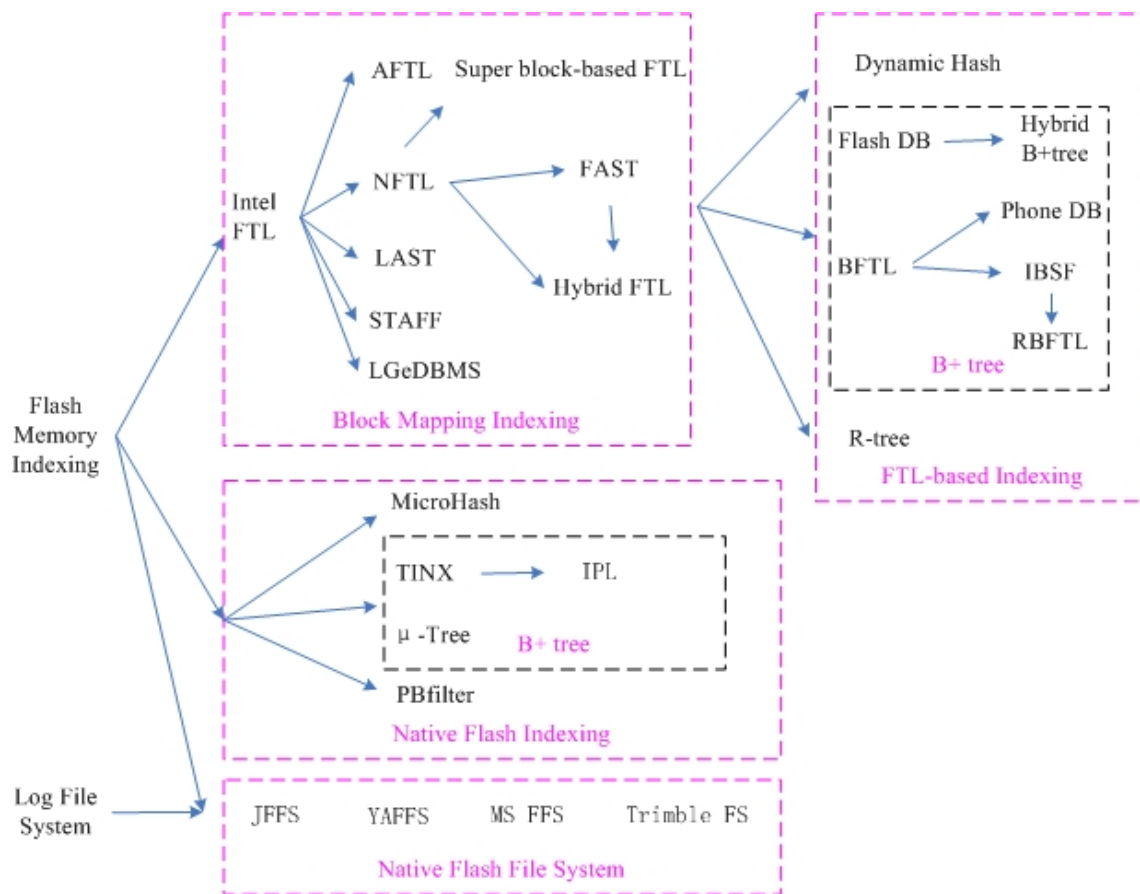


Outline

- Background
- Motivation
- **Related works**
- HF-Tree
- Experiments
- Conclusion



Related work

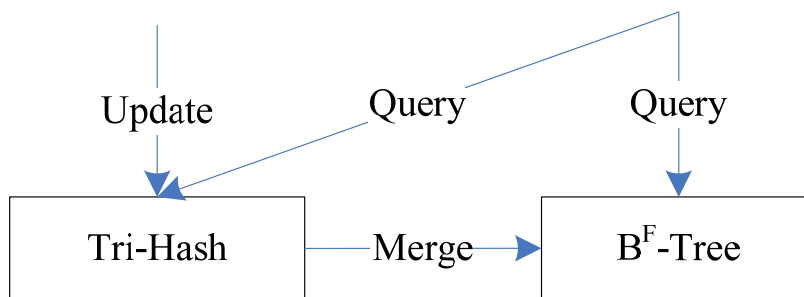




Outline

- Background
- Motivation
- Related works
- **HF-Tree**
- Experiments
- Conclusion

Framework

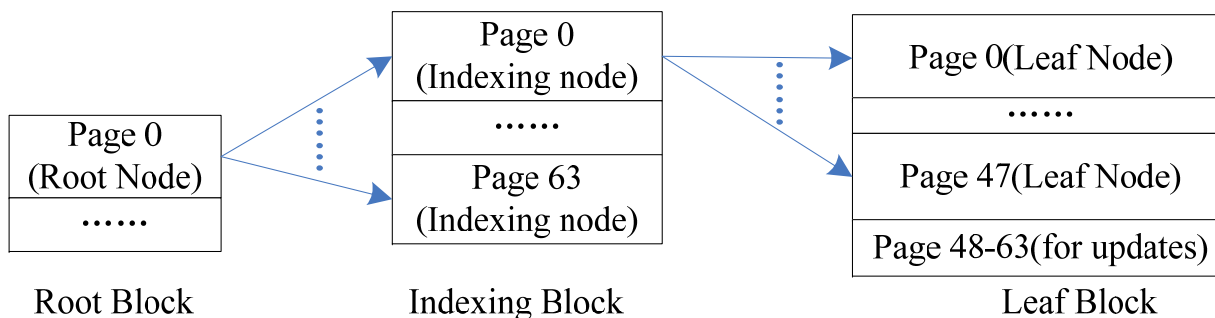


Architecture and operations of HF-Tree

HF-tree index is an integration of B^F-tree with Tri-hash. An update of index key is first stored in the Tri-hash structure, and will be merged into the B^F-tree only when necessary. A query will first look up the Tri-hash. If no result is obtained, the query will look up the B^F-tree. Insertions and deletions are recorded as updates and processed during index merge operations. In the following two subsections, we respectively explain the structures of B^F-tree and Tri-hash in detail.

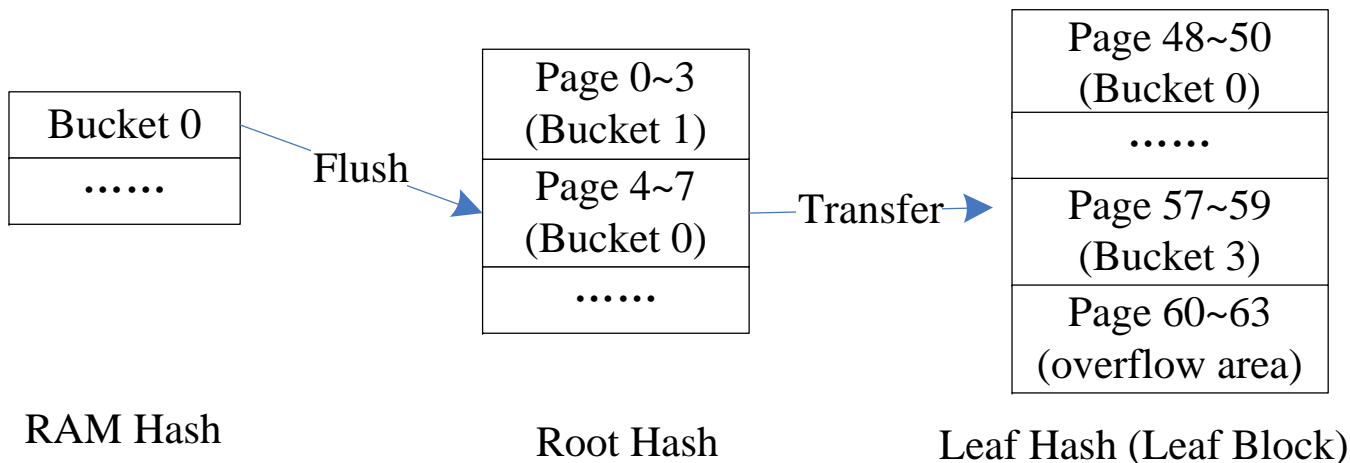


B^F-Tree



1. Different layers of B^F -tree nodes are stored in different types of blocks.
2. Different layers of B^F -tree employ different fanouts.
3. Coalesce and split are lazily performed and only executed during the merge operations of the leaf block.

Tri-hash

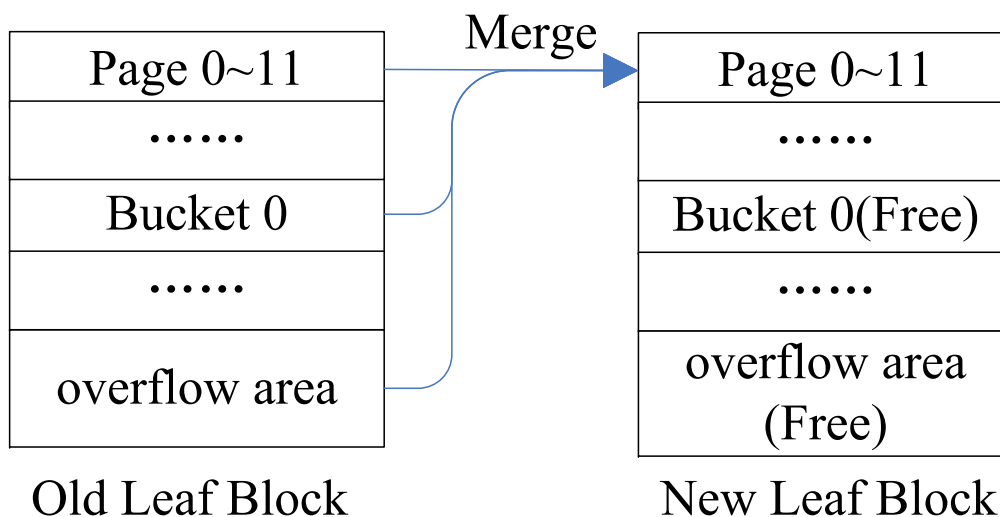


RAM Hash: Every bucket contains the updates of the keys falling in between two consecutive keys of the root node.

Indexing Hash: the indexing hash is used to store the updates from the RAM hash to defer updates further.

Leaf Hash: each leaf hash structure corresponds to a leaf block and is designed to defer the updates for the corresponding leaf block.

Integration of B^F -tree and Tri-Hash



When the leaf hash is full (i.e., the overflow area is filled up), merge integrates the updates into the B^F -tree by replacing the original keys with the updated ones in the leaf hash.

In the old leaf block, bucket 0 and the overflow area contain the updates for the first 12 pages. During the merge operation, updated keys are written to a new leaf node in a new leaf block and the leaf hash is initialized to be free in new leaf block.

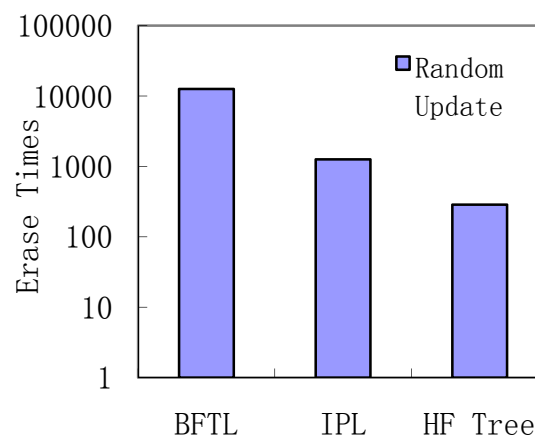
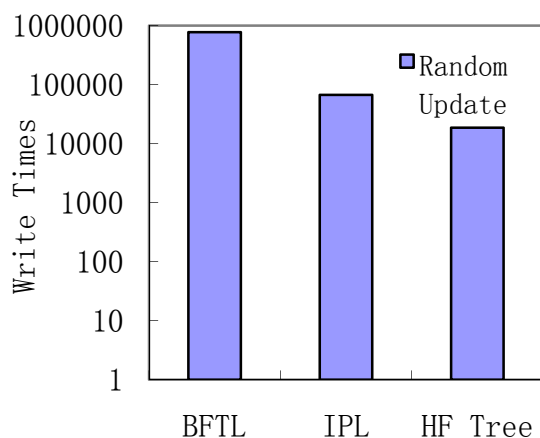


Outline

- Background
- Motivation
- Related works
- HF-Tree
- **Experiments**
- Conclusion

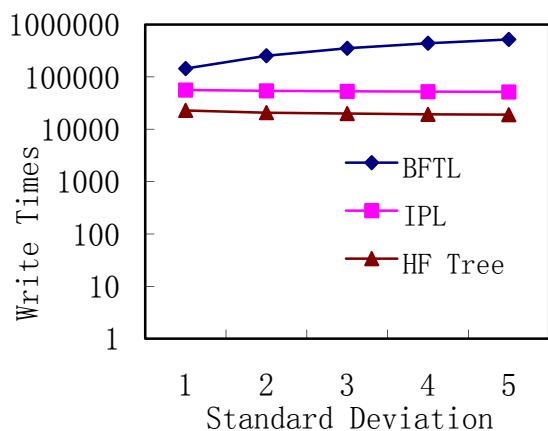


Random Update Model

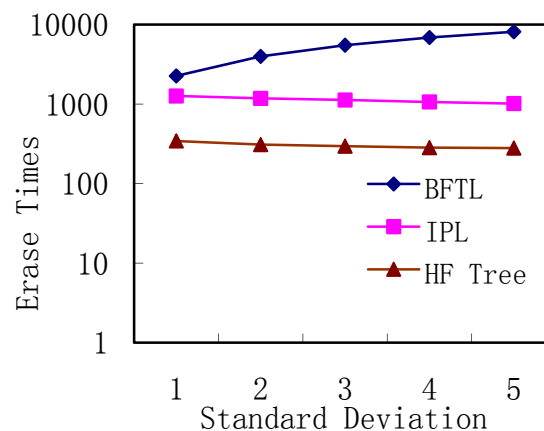


After 1,000,000 updates, we use the total number of page writes and total number of block erases to compare the update performance. While BFTL and IPL write flash pages 772,242 and 66,867 times, HF-Tree write only 18,664. Compared with BFTL and IPL, HF-Tree improves the performance by 97% and 72%. This validates that HF-tree can efficiently reduce writes by deferring and batching updates of index keys.

Normal Distribution Model



(c) Write performance of normal distribution

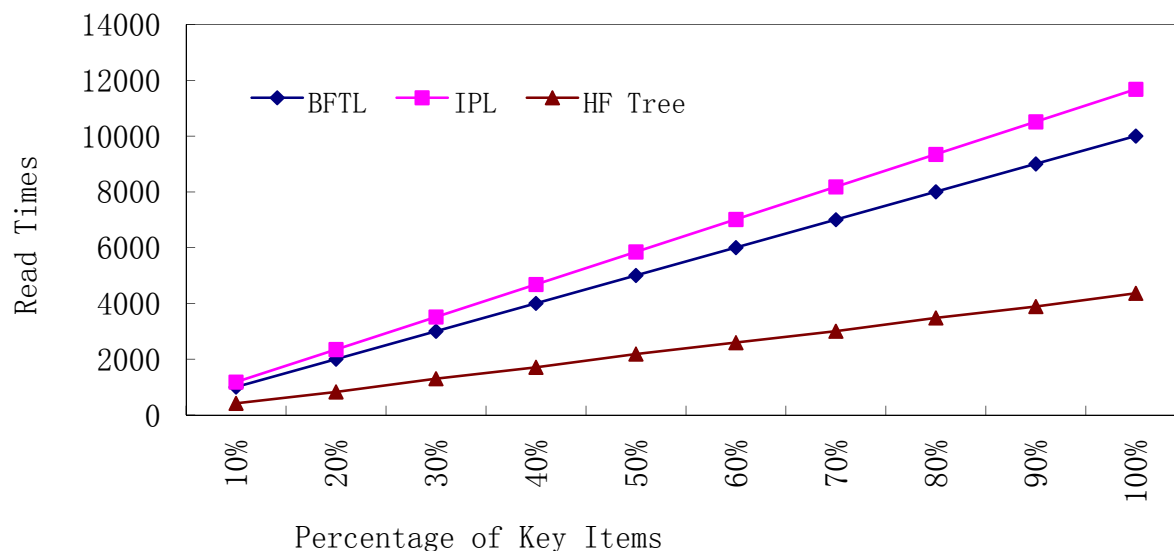


(d) Erase performance of normal distribution

The less the value of, the more skewed the update pattern. As shown in Fig. 7(a), when σ is set at 1, HF-tree improves the performance over BFTL and IPL by 84% and 60% respectively; when σ is set at 5, the improvement of HF-tree is slightly increased to 96% and 63% respectively. This proves that HF-tree can efficiently handle data updates with skewed update frequencies.



Range Query



Similarly, in order to support range queries, IPL must scan the log area frequently and repeatedly in order to get the latest data. Because HF-tree arranges the updated index keys in order by Tri-hash and B^F -tree, a range query can get the results by reading the data in the queried range only once.



Conclusion

- Updates are committed to flash memory and transferred among flash pages at the granularity of pages.
- Tri-hash improves the update performance by reducing the write cost and search time.
- Unlike log-based systems that need to exhaustively search logs to get the latest data, equality and range queries are efficiently improved by the integration of B^F -tree and Tri-hash.

Thank You 😊

Any Questions?