Solid State Drives and Databases

Chan Jie Ho
961948

Zhiping Liang
1071095

Geng Liu
1035248

Ran Lu
1200134

Introduction

- Architecture
- Hybrid Structure
- Indexing
- External Sorting Algorithms
- Query Processing
- Current Trends and Future Directions

Architecture - Hybrid Flash Memory SSD

Reduce Consumption

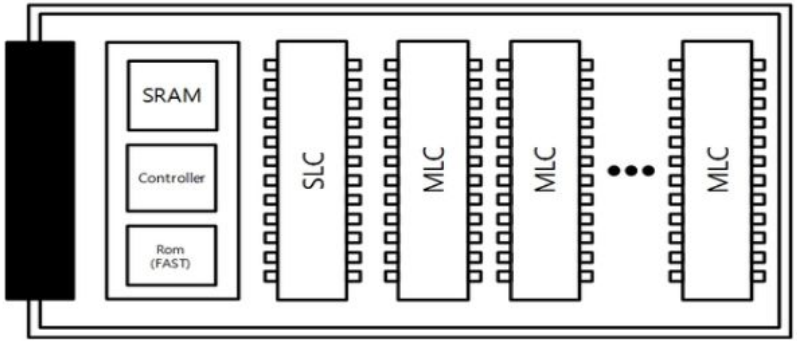


Fig 1. The structure of the Hybrid Flash Memory SSD

-Combines the advantages of SLC and MLC chips

-Faster I/O Speed

-Lower Price for bigger Storage space

Architecture - 2B-SSD

Reduce Consumption

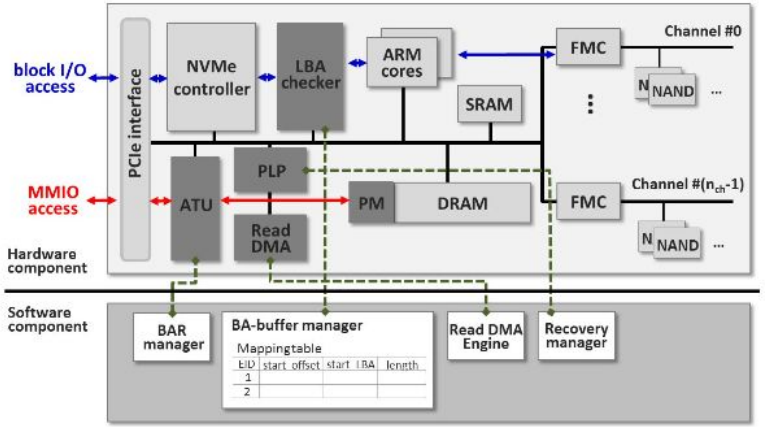


Fig 2. Overview of 2B-SSD

- More RAM, For better performance
- Higher throughput with no data loss

Architecture - Zero-Copying I/O Stack

Reduce latency

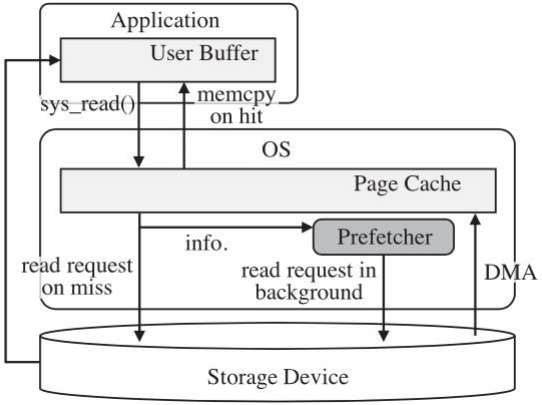


Fig 3. Overview of zero-copying I/O stack

-Take advantage of zero copy and page caching

-Higher hit rate and throughput

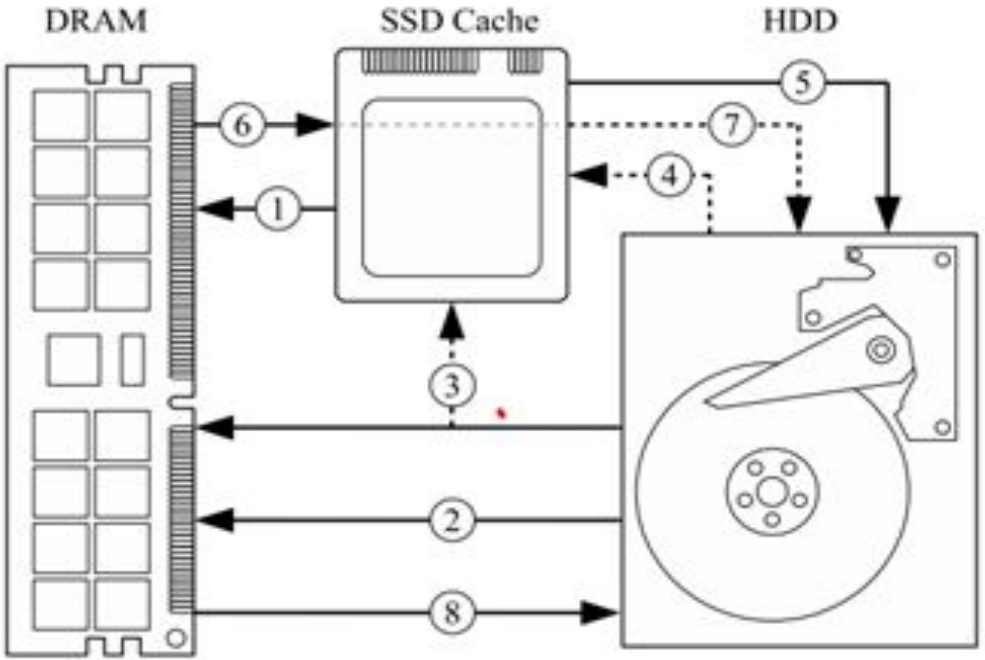
-Reduce I/O latency by up to 25%

Hybrid Structure

- **Caching Methods**
    1. SSD as Read-Only Cache
    2. SSD as Read-Write Cache
- **Tiering Methods**

SSD caching

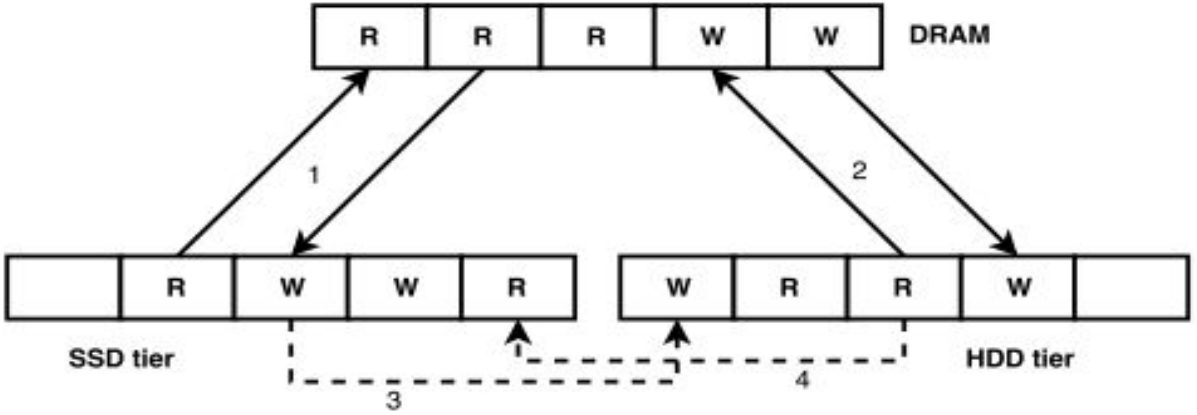
- **Read-Only Cache**
  - Write in HDD
  - Identify hot data
  - Move data to SSD
- **Read-Write Cache**
  - Write in SSD
  - Flush to HDD



Comparison of SSD Caching

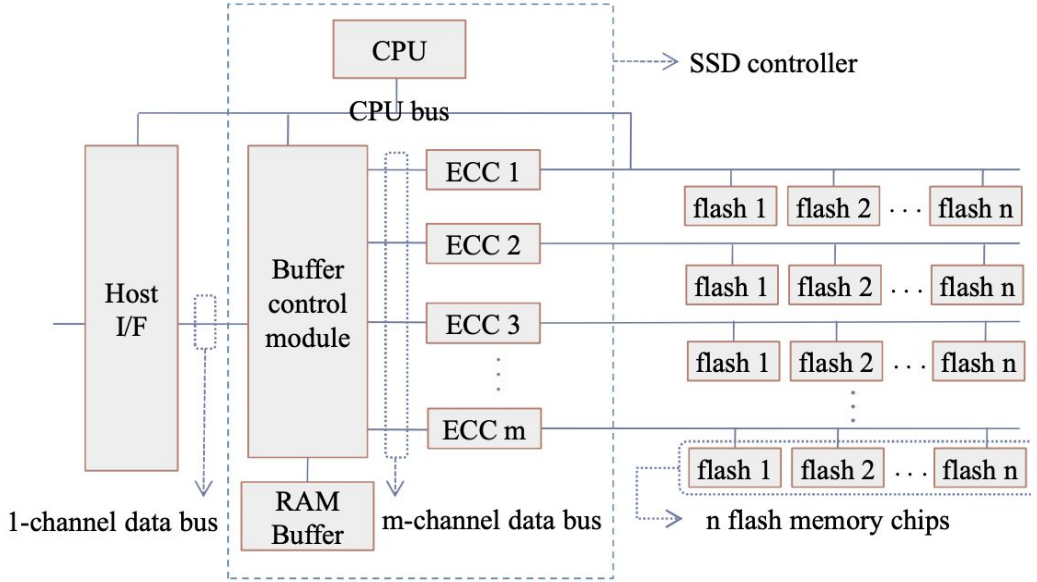
	Read-Only Cache	**Read-Write Cache**
Pros	Less write operation More cache space for read data	Data can be kept in SSD for a long time before the space is full
Cons	Need good replacement algorithm Garbage Collection	Much more write operation Data synchronization problem

SSD tiering



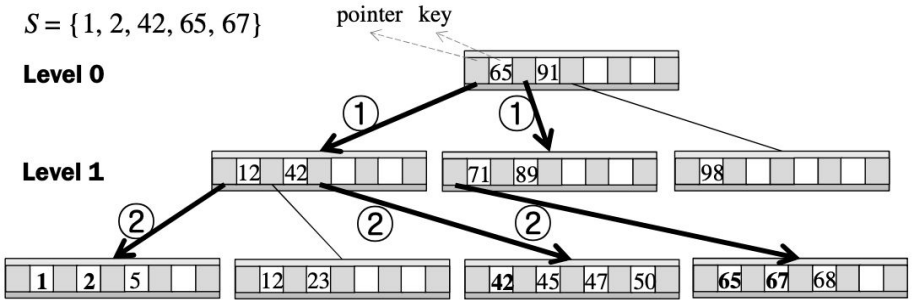
- Data store in either SSD or HDD
- Can have multiple tiers
- Less data relocation
- Slower to optimize the system

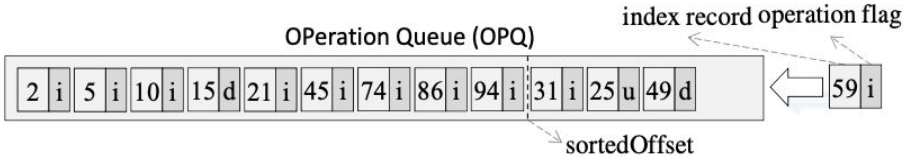
Indexing - Parallelism in FlashSSDs



- Channel-level parallelism
- Package-level parallelism
- Performance enhancement by multiplying **m** channels and **n** flash chips

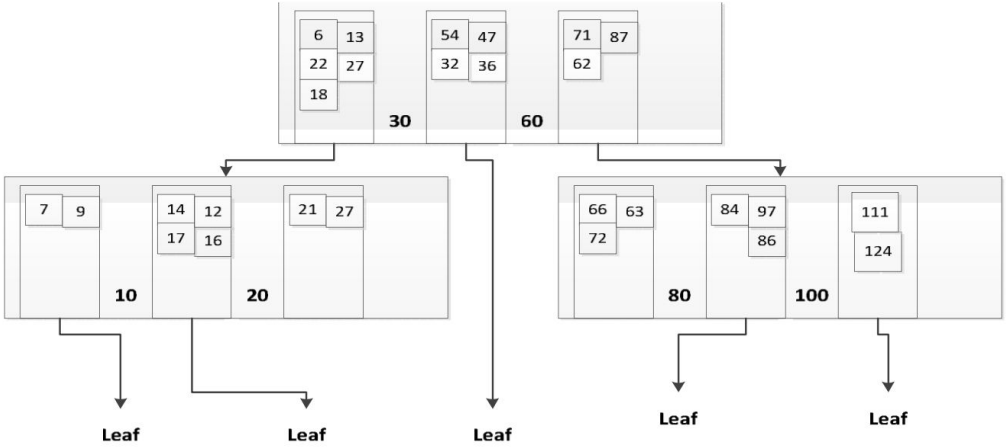
Indexing - PIO-Tree





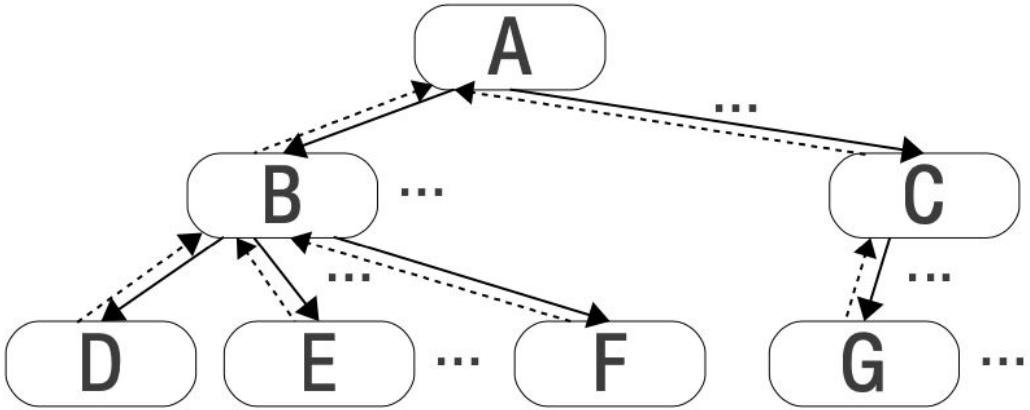
- Look at entries in multiple nodes through parallel synchronous I/O
- A enlarged resizing unit called Leaf Segment
- Append an index entry right next to the most recently inserted operation queue, same for update and delete

Indexing - AB-Tree



- Insert real data in both internal and leaf nodes and put data entries into buckets in each node
- Each bucket has a data part and a buffer part
- A modified node structure contains a pointer list, a key list and a bucket list
- Each bucket has a threshold value

Indexing - FB-Tree



- Abandon the sibling pointers
- Maintain dash pointers to parents' in-memory buffer
- If a node splits or merges, pointers to the siblings are acquired through its parent

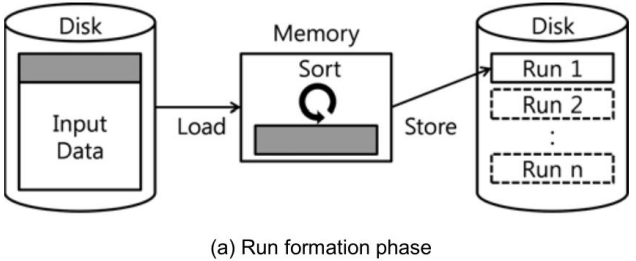
Indexing - Comparison

B+Tree Variants	PIO-Tree	AB-Tree	FB-Tree
Optimisation	Multiple simultaneous search requests and optimised node size	Put data entries into buckets in each node for bulk insertion and update	Break sibling pointers in leaf nodes and maintain dash pointers to parents
Pros	Reduce average write cost from the number of pages / 2 to 1 and shorten search time	Insert many entries via bulk insertion and achieve the exact I/O cost as of inserting one entry	Each update is independent without rewriting the whole tree.
Cons	Larger nodes enhance FlashSSD bandwidths, but the latency of each I/O operation also increases.	It is challenging to decide the ratio of the buffer part of buckets for arbitrary workloads.	A large group of such writes cannot be collected into a single large write and cause problems.

External Sorting Algorithms – MergeSort

External Sorting Algorithms

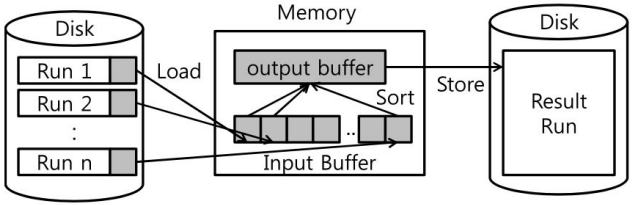
- Necessary when data volume >>> main-memory capacity.
- They revolve around runs
- Two phases: Run generation phase and run merge phase
- Performance is highly dependent on their I/O cost



(a) Run formation phase

MergeSort

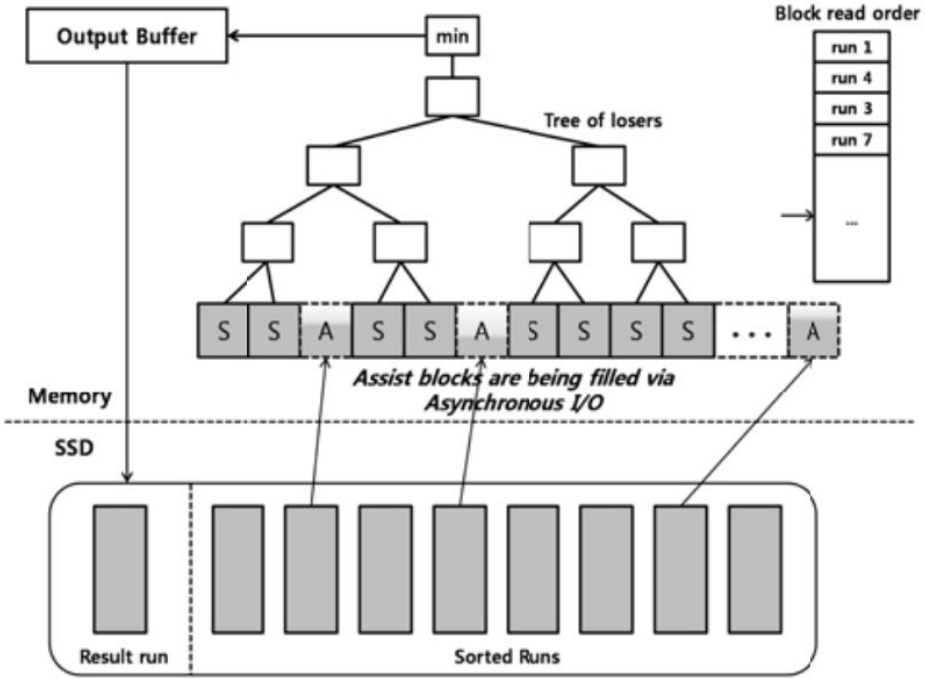
- Sorts them using buffers in the merge phase
- Very I/O Intensive



(b) Merge phase

Flash MergeSort (FMSort)

- Improves on merge phase
- Utilises high random I/O bandwidths of SSDs (exploiting internal parallelism) to reduce I/O operations
- Multiple data blocks are read simultaneously into main memory via multiple asynchronous I/Os
- Block read order calculated during generation phase by sorting each data block's first tuple's key



FMSort (cont.)

✓ Outperformed traditional MergeSort by up to 4.87 times

✓ Outperformed MergeSort with double buffering up to 4.67 times

✗ Only focused on improving the merge phase

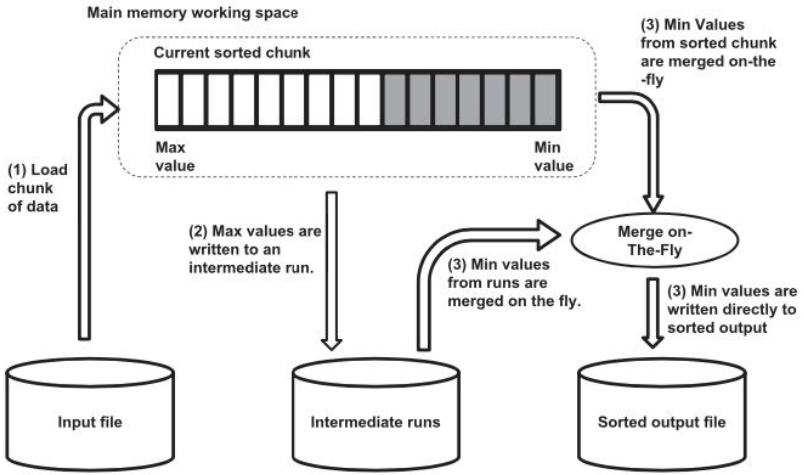
✗ Incurs additional overhead in the generation phase

- Only sequential reads and writes executed in generation phase
- Leverage on short access latency during merge phase

Merge ON-the-Run External Sorting (MONTRES)

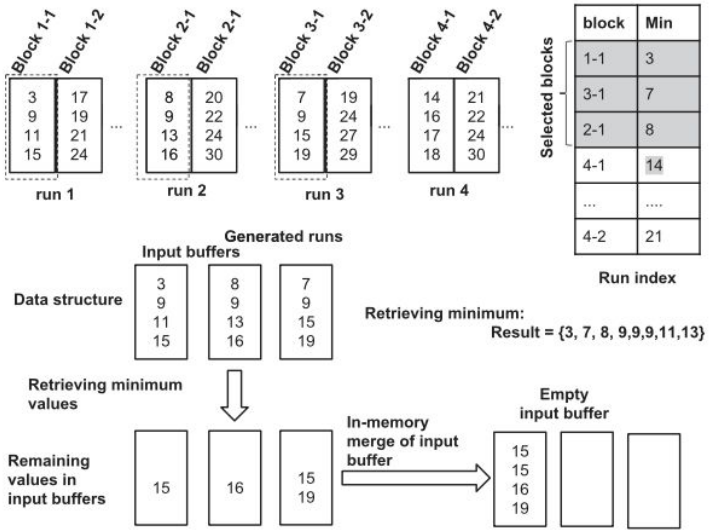
- Optimising both phases at the cost of a small number of extra read operations

- Use random reads on SSDs to select blocks in ascending order
- Merge on-the-fly mechanism that evicts small values to the sorted file all through the phase
- Adopt continuous run expansion policy to create the largest possible runs



MONTRES (cont.)

- Perform merge phase in one pass by continuously retrieving minimum values from generated runs



✓ Outperformed traditional algorithms by reducing execution times by >40% when file size to main-memory size ratio is large

✓ Continuous run expansion policy is efficient when data is partially sorted.

ActiveSSDs and ActiveSort

- Offload parts of data processing process to Active SSD to improve I/O throughput and remove extra data transfer
- Postpone merge phase until the host issues read requests to the final sorted output
- Data is merged on-the-fly inside active SSD and result is returned to the host

✓ Easily integrated with host applications
✓ Allows for concurrent computation and processes I/Os efficiently
✓ 36.1% improvement in Hadoop applications
✓ 40.4% reduction in writing operations

✘ Poor performance on small-sized records and when using fixed sizes for keys and records
✘ Requires an interface to enable the data to merge on-the-fly inside the SSD

ActiveSort vs MergeSort

Method	Read (MB)	Write (MB)
ActiveSort	2048.3	2048.4
Merge sort	4085.9	4096.7

- ActiveSort transfers read and write data == data set size
- MergeSort generates more – merge phase transfers data between main memory and disk.

Query Processing – Join Queries

- Prolong SSD lifetimes by reducing number of temporary write operations in query processing operations
- Joining large relations requires writing intermediate results to SSDs
- Hybrid Hash Join (HHJ) and Sort-Merge Join (SMJ) produce temporary writes

DigestJoin

- Exploiting random reads in SSDs
- Join on digest tables to reduce the size of the temporary writes

Advanced Block Nested Loop Join (ANLJ)

- Extension of BNLJ
- HHJ and SMJ outperforms BNLJ, but BNLJ doesn't produce temporary writes
- Changes how the outer relation is set and how the tuples are read in the input buffer

DigestJoin vs ANLJ

Join Variants	*DigestJoin*	*ANLJ*
Pros	Outperforms SMJ in various system config.	Outperforms SMJ, GHJ, HHJ if relations partially-sorted by join-keys Performance on par if buffer size sufficiently large Eliminates temporary writes and significantly improves SSD lifetime
Cons	Need to fetch original tuples that satisfy join Page fetching will greatly increase number of read operations	Underperforms if buffer size is not large enough and not sorted

Current Trends and Future Directions

- More efficient Algorithm
- New Hybrid Structure SSD
- Reducing Consumption
- New material

THE UNIVERSITY OF MELBOURNE

Q & A

THE UNIVERSITY OF MELBOURNE

Reference List

- Akritidis, L., Bozanis, P., Fevgas, A., & Manolopoulos, Y. (2018, December). Indexing in flash storage devices: a survey on challenges, current approaches, and future trends. https://doi.org/10.1007/s00778-019-00559-8
- Bae, D., Jo, I., Choi, Y., Hwang, J., Cho, S., Lee, D., & Jeong, J. (2018). 2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives. 2018 ACM/IEEE 45Th Annual International Symposium On Computer Architecture (ISCA). https://doi.org/10.1109/isca.2018.00043
- Hoseinzadeh, M. (2019). A Survey on Tiering and Caching in High-Performance Storage Systems. *ArXiv:1904.11560 [Cs].* https://arxiv.org/abs/1904.11560
- Jiang, Z., Li, C., Wu, Y., Xing, C. & Zhang, Y. (2014, September). AB-Tree: A Write-optimized Adaptive Index Structure on Solid State Disk. https://ieeexplore.ieee.org/abstract/document/7058011
- Jørgensen, M. V., Rasmussen, R. B., Šaltenis, S. & Schjønning, C. (2011, September). FB-tree: a B+-tree for flash-based SSDs. https://dl.acm.org/doi/10.1145/2076623.2076629
- Kim, S., Lee, G., Woo, J., & Jeong, J. (2021). Zero-Copying I/O Stack for Low-Latency SSDs. IEEE Computer Architecture Letters, 20(1), 50-53. https://doi.org/10.1109/lca.2021.3064876
- Kim, S., Lee, S., Park, S., Roh, H. & Shin, M. (2011, December). B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives. https://dl.acm.org/doi/10.14778/2095686.2095688
- Laga, A., Boukhobza, J., Singhoff, F., & Koskas, M. (2017). MONTRES : Merge ON-the-Run External Sorting Algorithm for Large Data Volumes on SSD Based Storage Systems. IEEE Transactions on Computers, 66(10), 1689–1702. https://doi.org/10.1109/TC.2017.2706678
- Lee, J., Roh, H., & Park, S. (2016). External Mergesort for Flash-Based Solid State Drives. IEEE Transactions on Computers, 65(5), 1518–1527. https://doi.org/10.1109/tc.2015.2451631
- Nam, B., Na, G., & Lee, S. (2010). A Hybrid Flash Memory SSD Scheme for Enterprise Database Applications. 2010 12Th International Asia-Pacific Web Conference. https://doi.org/10.1109/apweb.2010.70
- Niu, J., Xu, J., & Xie, L. (2018). Hybrid Storage Systems: A Survey of Architectures and Algorithms. *IEEE Access, 6, 13385–13406.* https://doi.org/10.1109/access.2018.2803302