

Updatable Learned Indexes Meet Disk-Resident DBMS - From Evaluations to Design Choices

Hai Lan¹, Zhifeng Bao¹, J. Shane Culpepper¹, Renata Borovica-Gajic²

¹ RMIT University

² The University of Melbourne



Motivation

Promising Performance of Learned Indexes

(compared to B+-tree)

3x performance

10x smaller index size

Widely Used Disk-based DBMSs

Indexes' size can be quite large

Main memory is a crucial resource

Existing Studies on Disk

Work on LSM Tree [1, 2]

Tend to suffer from a bad read performance

Can the idea of learned index benefit the on-disk updatable indexing techniques, with the hope of uprooting B+-tree?

[1] Hussam Abu-Libdeh, et al. 2020. Learned Indexes for a Google-scale Disk-based Database. CoRR abs/2012.12501 (2020)

[2] Yifan Dai, et al. 2020. From WiscKey to Bourbon: A Learned Index for Log-Structured Merge Trees. In OSDI. 155–171

Questions

For the first time, we develop the on-disk version of the typical in-memory learned indexes and conduct a comprehensive evaluation.

Q1 - How good are updatable learned indexes when compared to a B+-tree?

Q2 - How much storage do learned indexes require on disk?

Q3 - What impact do different **block sizes** have on performance?

Q4 - Do the learned indexes show consistent performance among different datasets when disk-resident?

Q5 - What impact do the **buffers** have?

Representative Learned Indexes' Design


gapped array 
 packed array 

Table 1: A Taxonomy of Studied Indexes

Index	Year	Inner Node		Leaf Node		Insert ²	Data Partition	Node Size	Structure Modification	
		Search Algo. ¹	Error	Search Algo.	Error				Strategy	Memory Reuse
B+-tree	-	B.S	Tunable	B.S	Tunable	Empty Slot	Evenly	Tunable	Greedy	✓
FITing-tree [1]	2019	B.S	Tunable	Model + B.S	Tunable	Buffer	Greedy	Node-related	Greedy	×
PGM [2]	2020	Model + B.S	Tunable	Model + B.S	Tunable	Append/Rebuild	Streaming Algo.	N.A.	Greedy	×
ALEX [3]	2020	Model	Exact	Model + E.S	Unfixed	Gapped Array	-	Tunable	Cost-based	×
LIPP [4]	2021	Model ³	Exact	Model	Exact	Gapped Array	-	Tunable ⁴	Greedy	×

¹ B.S, E.S, and Model denote binary search, exponential search, and predicting the position with a model, respectively.

² Here, we refer to how these indexes store the new insertion key-value pairs.

³ LIPP does not distinguish the inner node and leaf node. We set the *Inner Node* and *Leaf Node* with same values.

⁴ Although the authors state that a maximum node size is set, they could create a larger node size than the parameter (line 2 in Algorithm 5 [30]).

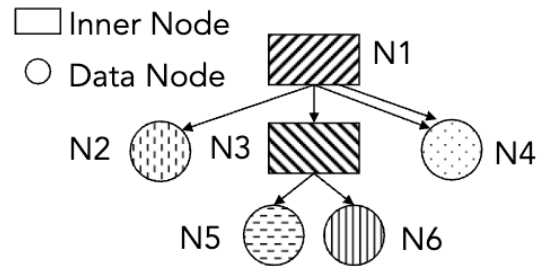
[1] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, Tim Kraska: FITing-Tree: A Data-aware Index Structure. SIGMOD 2019.

[2] Paolo Ferragina, Giorgio Vinciguerra: The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. VLDB 2020.

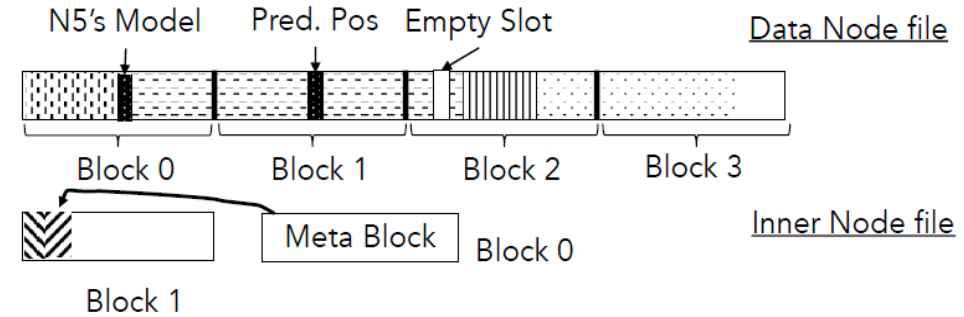
[3] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, Tim Kraska: ALEX: An Updatable Adaptive Learned Index. SIGMOD 2020

[4] Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, Chunxiao Xing: Updatable Learned Index with Precise Positions. VLDB 2021.

Implementation on Disk



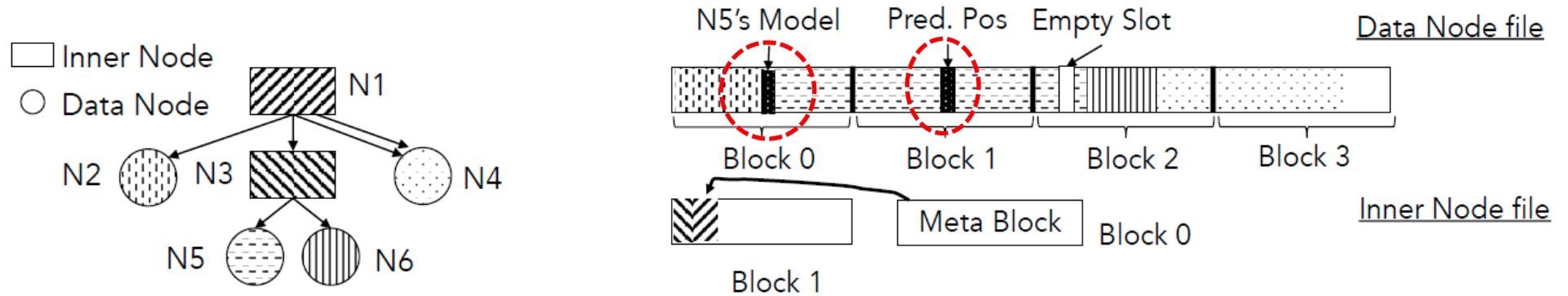
(a) ALEX Structure



(b) ALEX Layout on Disk

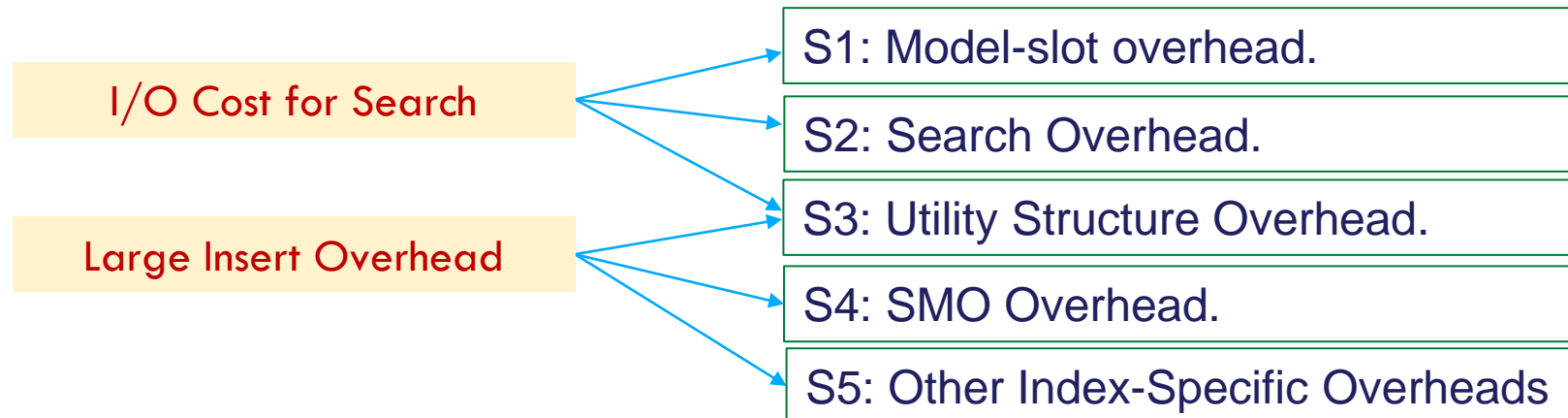
- ALEX
 - Meta block pointing to the root node.
 - Two separated files for inner nodes and data nodes.
 - Store the data continuously
- FITing-tree
 - Adopt the segmentation algorithm in PGM to partition the data
 - For scan operation, store the physical addresses of siblings for each leaf node, which is same as a B+-tree
 - Introduce an extra block to hold the keys that are smaller than current domain.
- LIPP
 - Replace the bitmap (to indicate the slot types) in LIPP with the flag in each item.

From Memory-Resident to Disk-Resident



(a) ALEX Structure

(b) ALEX Layout on Disk



Evaluation – Setup

- Dataset statistics
 - Number of segments under a given error bound
 - Conflict degree

Table 3: Dataset Profiling under Error Bound and Conflict Degree (block size = 4 KB)

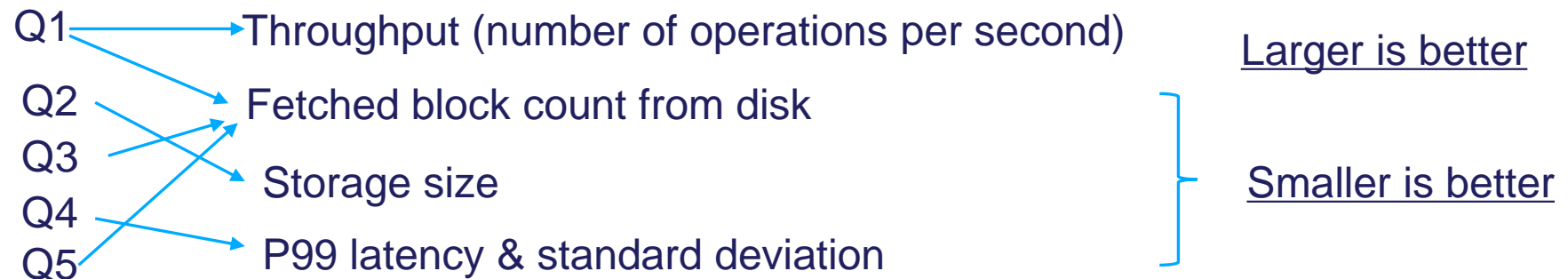
Error	YCSB	FB	OSM	Covid	History	Genome	Libio	Planet	Stack	Wise	OSM(800M)
16	70,135	2,120,485	1,351,170	231,852	303,737	3,153,966	291,257	1,416,012	54,073	246,463	6,175,387
64	6,952	523,006	326,932	42,695	40,817	295,604	77,401	268,247	6,956	27,553	1,375,143
256	23	119,891	81,392	8,630	8,464	23,228	19,333	55,061	950	4,713	328,623
1024	1	18,495	20,925	1,890	2,029	4,975	3,616	12,001	196	1,184	81,577
B+-tree	980,393	980,393	980,393	980,393	980,393	980,393	980,393	980,393	980,393	980,393	3,921,569
Conflict Degree	4	114	4,106	27	9	585	2	22	1	10	10,107

Evaluation – Setup

- Workload Design

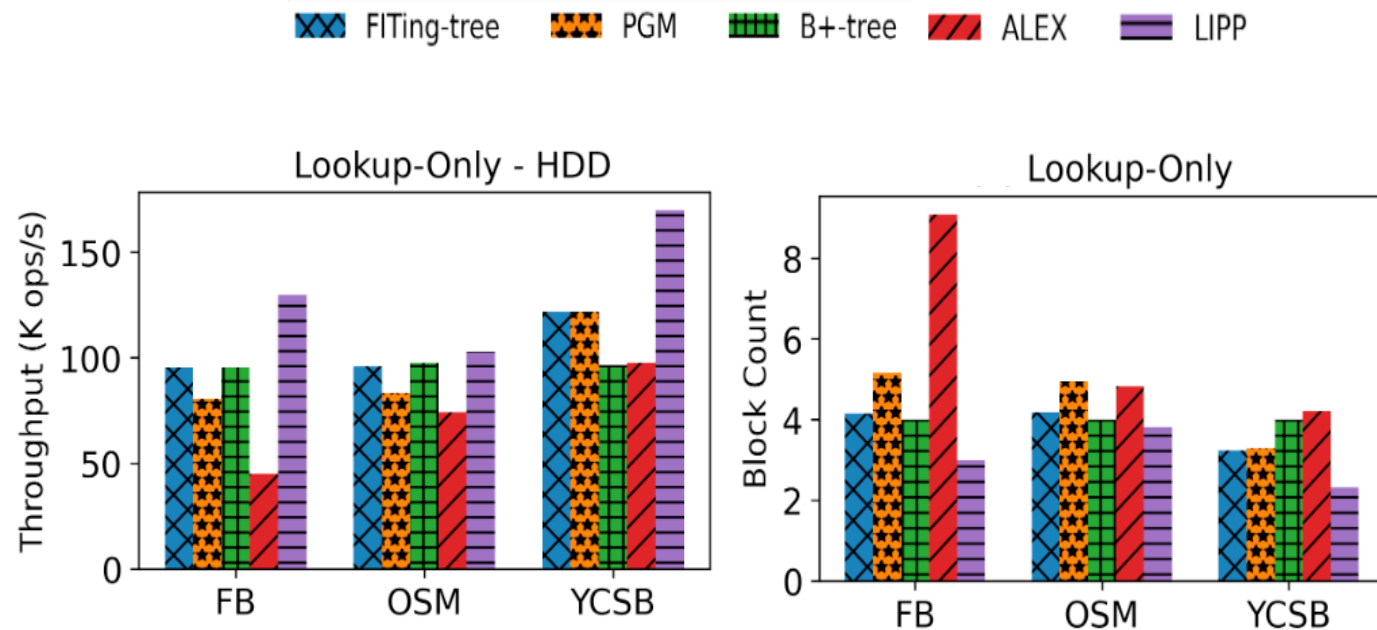
Lookup-Only	Scan-Only	Write-Only	Write-Heavy	Balanced	Read-Heavy
100% lookups	100% scans	100% inserts	90% inserts 10% lookups	50% inserts 50% lookups	10% inserts 90% lookups

- Performance Metrics



Evaluation – Q1: How Good Compared to B+-tree

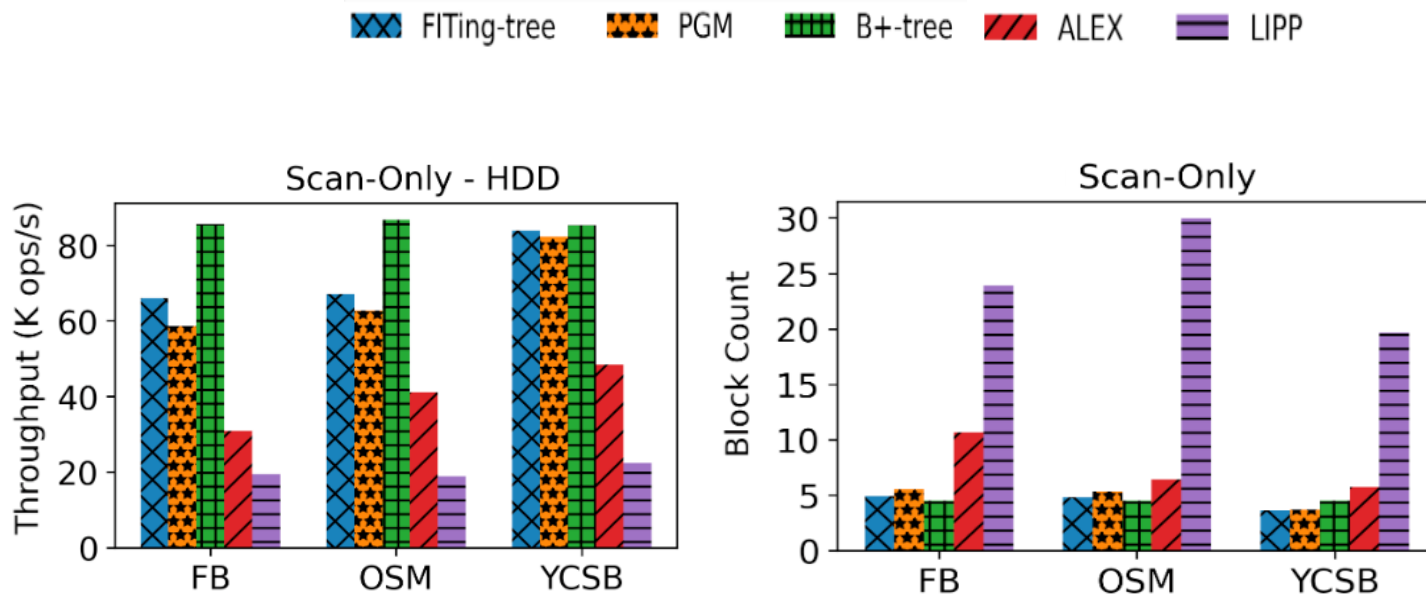
- Lookup-Only Workload



- ✓ The throughput of these indexes is determined by the number of fetched blocks from disk.
- ✓ Most existing learned indexes are competitive or outperforming B+-tree.
- ✓ B+-tree exhibits stable performance, while the performance of the learned indexes fluctuates.

Evaluation – Q1: How Good Compared to B+-tree

- Scan-Only Workload



- ✓ Regardless of the dataset hardness, B+-tree outperforms others across all datasets.
- ✓ ALEX and LIPP exhibit the worst performance in the Scan-Only workload.

Evaluation – Q1: How Good Compared to B+-tree

- Fetched Block Breakdown for Read-Only Workload

	FB				OSM				YCSB			
	FITing-tree	PGM	ALEX	LIPP	FITing-tree	PGM	ALEX	LIPP	FITing-tree	PGM	ALEX	LIPP
Inner Node Count	3	5	6.7	1.8 (18.8)	3	5	2.7	2.3 (23.1)	2	3	3	1.3 (16.7)
Inner Block Count	3	3.9	6.5	-	3	3.7	2.6	-	2	2	2.2	-
Leaf Block Count (Lookup)	1.2	1.3	2.6	3.0	1.2	1.2	2.2	3.8	1.2	1.3	2	2.3
Leaf Block Count (Scan)	2	1.7	4.1	24.0	1.8	1.5	3.8	30.0	1.6	1.7	3.6	19.7

- ✓ ALEX and PGM can store more than one inner nodes into one block. Thus, the number of fetched blocks is smaller than the number node.
- ✓ ALEX fetches more blocks on leaf nodes for scan due to the need for bitmap access.
- ✓ LIPP fetches much more nodes and blocks for scan queries due to its single node type design.

Evaluation – Q1: How Good Compared to B+-tree

- How Good A Hybrid Design, where we adopt a B+-tree styled leaf nodes?

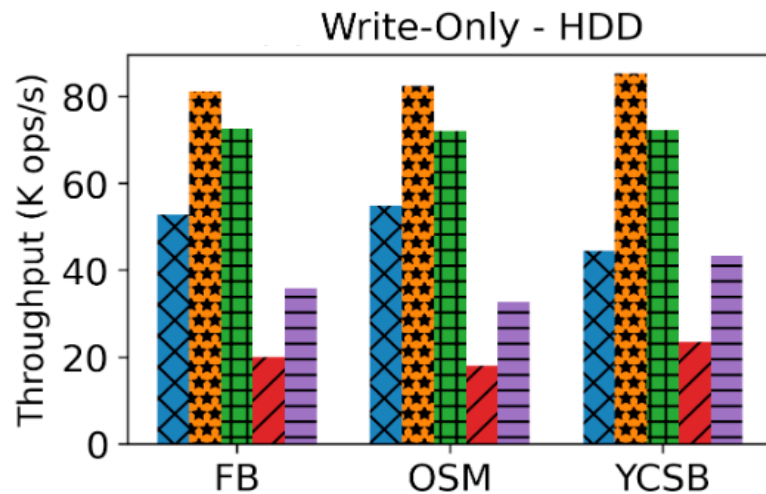
	FITing-Tree	PGM	ALEX	LIPP	B+-tree
FB	3.25/3.74	3.25/3.74	4.02/4.51	3.15/3.64	4.0/4.49
OSM	4.25/4.74	4.17/4.66	4.77/5.26	4.5/5.0	4.0/4.49
YCSB	3.25/3.74	3.25/3.74	4.0/4.49	3.01/3.5	4.0/4.49

- ✓ These learned indexes can achieve similar or better performance than a B+-tree.
- ✓ The fetched blocks of LIPP and ALEX on scan-only workload are reduced significantly.

Evaluation – Q1: How Good Compared to B+-tree

- Write-Only Workload

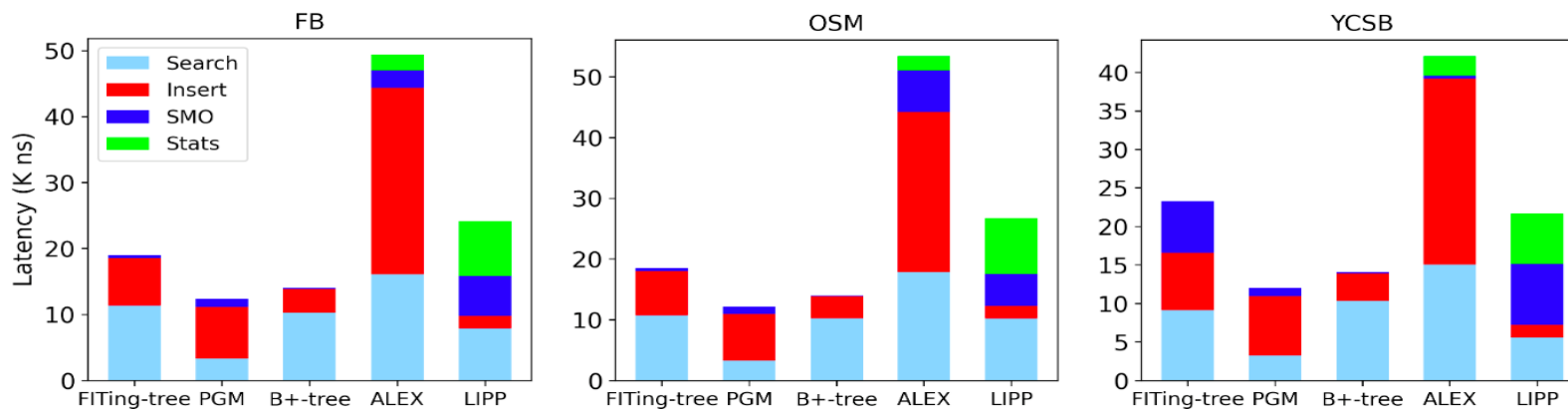
Legend: FITing-tree (blue cross-hatch), PGM (orange stars), B+-tree (green grid), ALEX (red diagonal), LIPP (purple horizontal lines)



- ✓ The relative ranking of all indexes is consistent across all datasets, with PGM significantly outperforming other methods.
- ✓ Other than PGM, B+-tree significantly outperforms other learned indexes.

Evaluation – Q1: How Good Compared to B+-tree

- Write-Only Workload

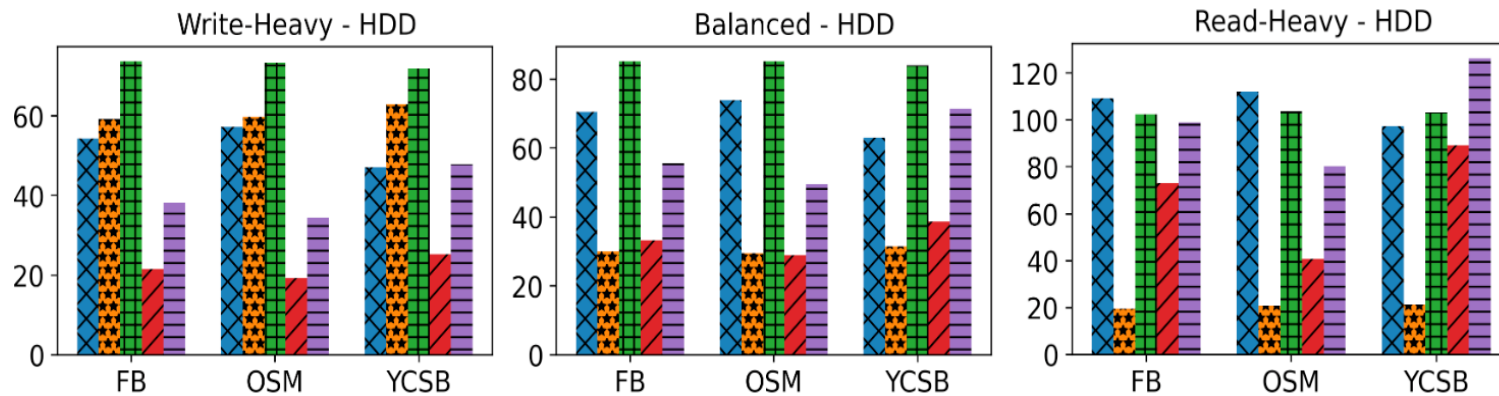


- ✓ PGM has the smallest search overhead.
- ✓ FITing-tree has the similar search overhead with a B+-tree while a large insertion overhead.
- ✓ ALEX shows a large insertion overhead.
- ✓ LIPP has a large maintenance overhead and even larger than ALEX.

Evaluation – Q1: How Good Compared to B+-tree

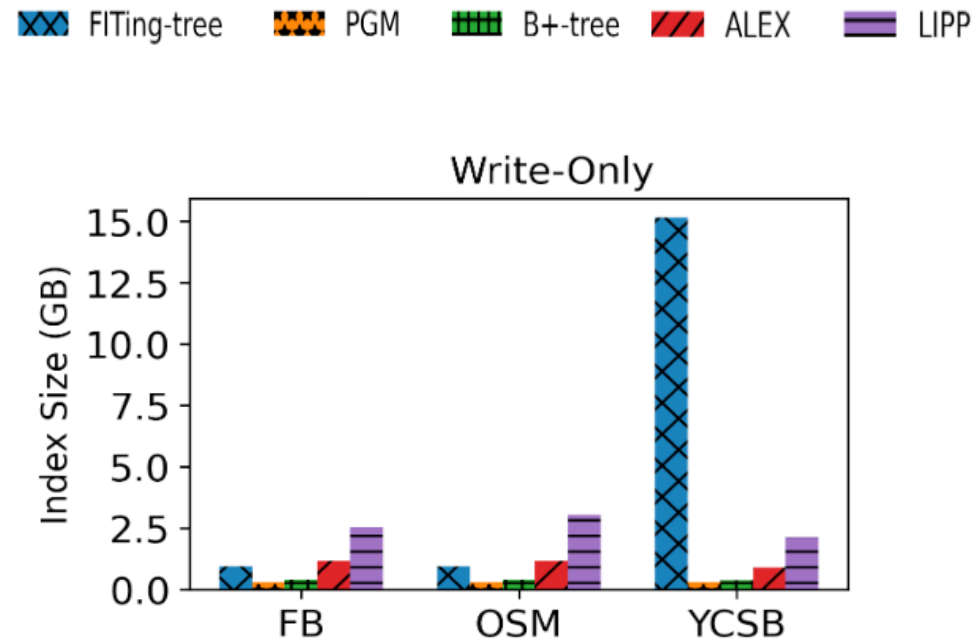
- Mixed Workload

FITing-tree PGM B+-tree ALEX LIPP



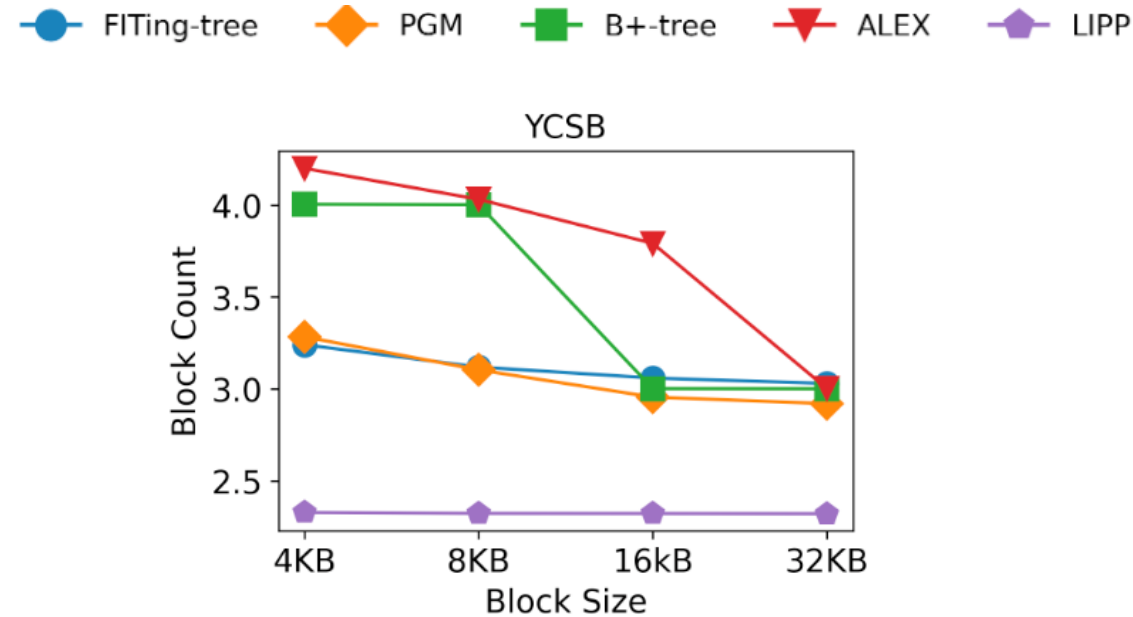
- ✓ In most cases (6 out of 9), the B+-tree outperforms learned indexes.
- ✓ B+-tree and PGM exhibit consistently performance among different datasets while the performance of other learned indexes fluctuates significantly.
- ✓ As the read ratio increases, the throughput of PGM degrades severely, unlike the alternatives where it increases.

Evaluation – Q2: Storage Usage



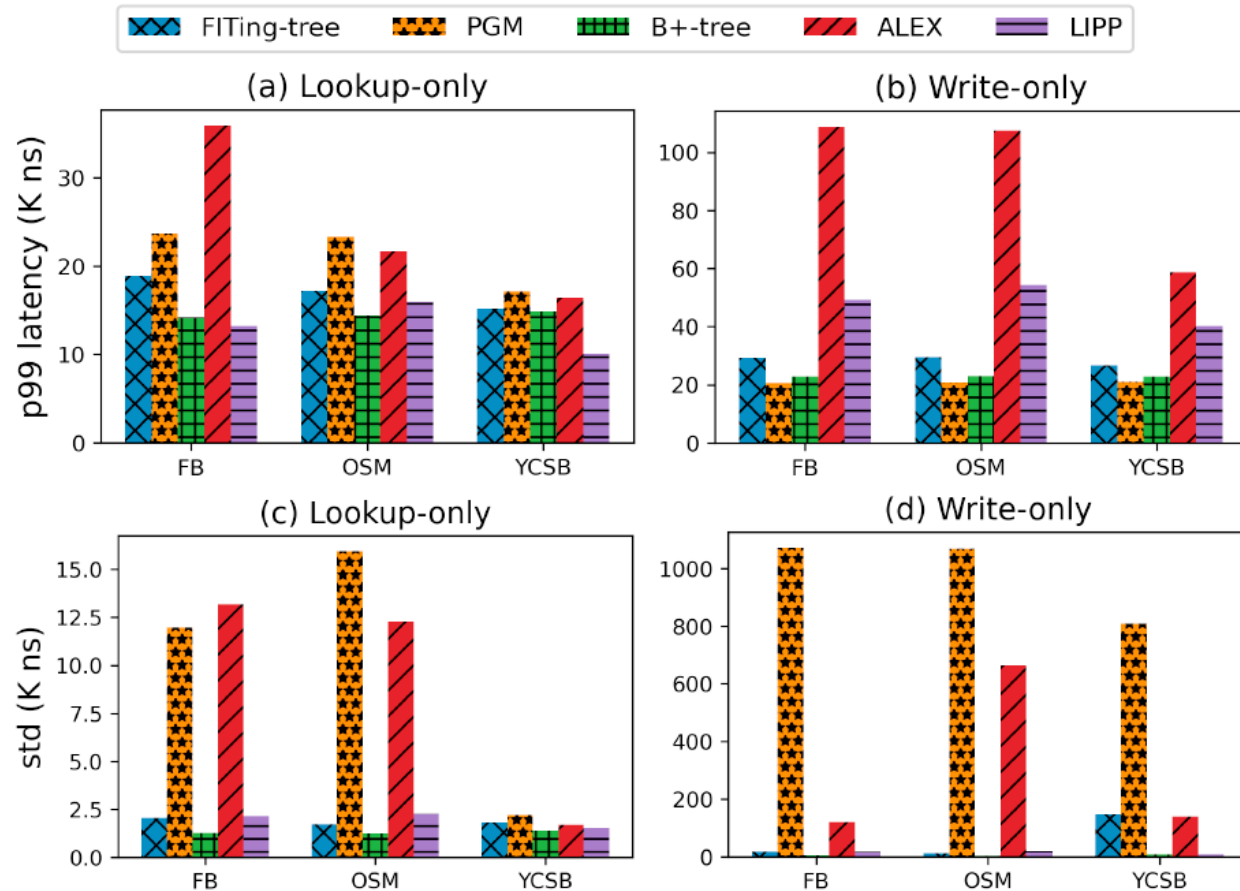
- ✓ Learned indexes usually require more storage space than B+-tree.
- ✓ The index sizes of FITing-tree and LIPP highly depend on the distribution of the dataset indexed.

Evaluation – Q3: Impact of Block Size



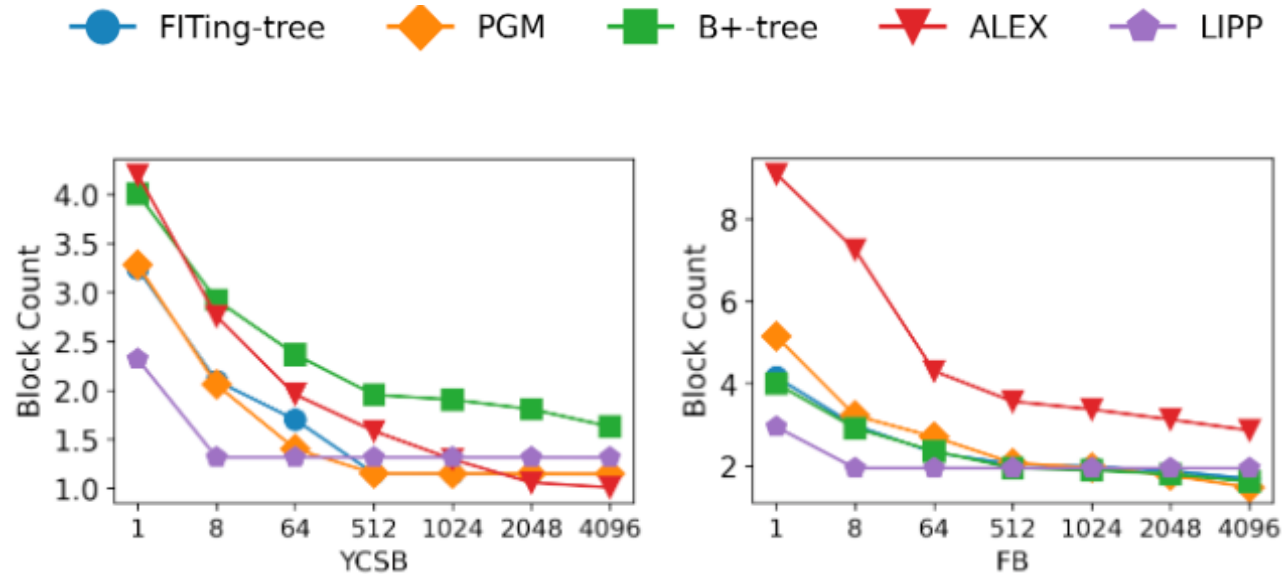
- ✓ The block size variation has different impacts on different Indexes. The number of blocks fetched by LIPP does not change when the block size is varied while other indexes tend to fetch fewer blocks with a large size.

Evaluation – Q4: Robust Performance Profile



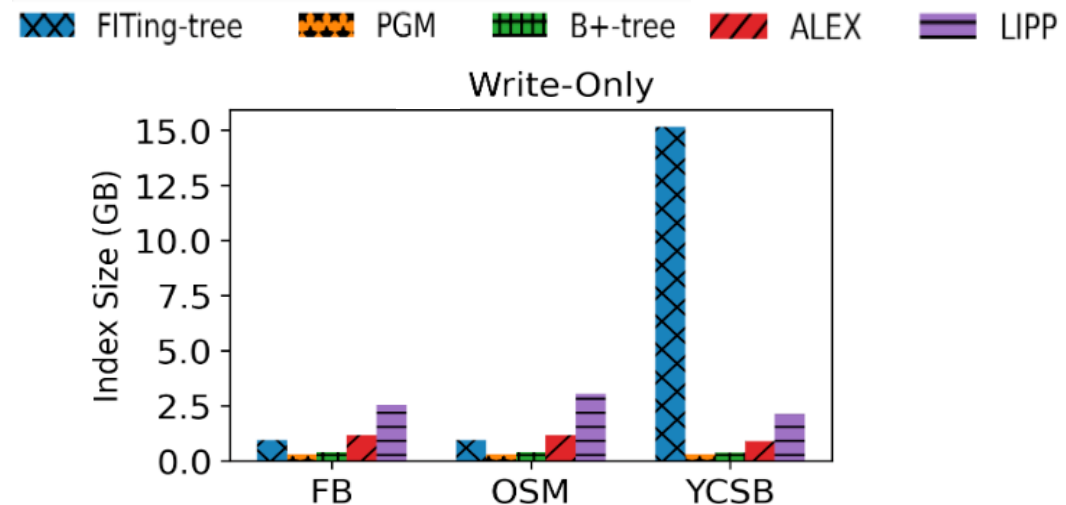
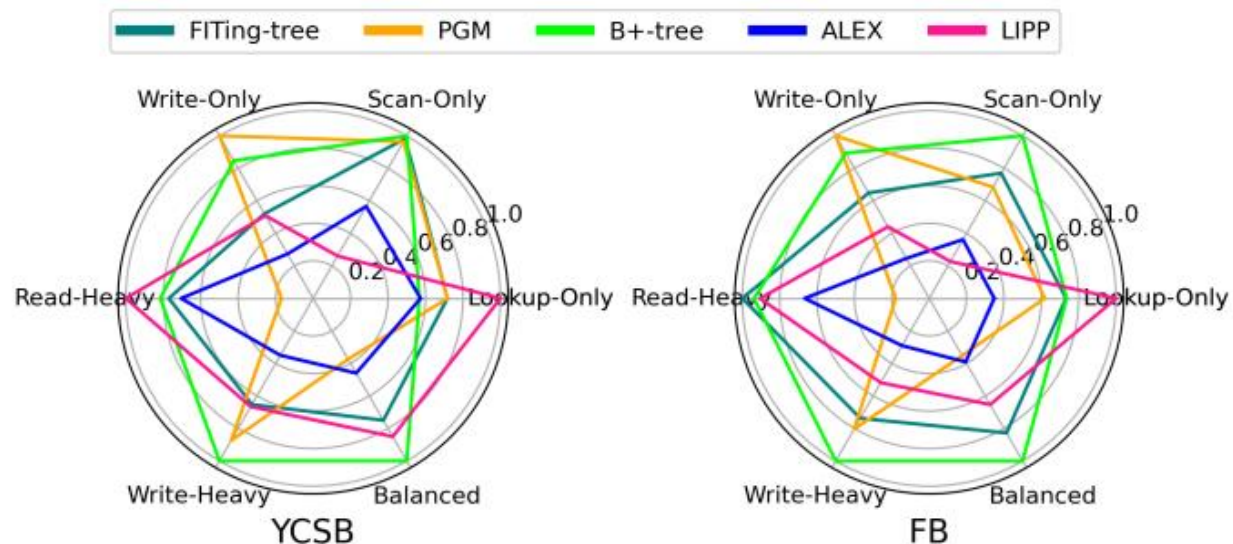
- ✓ Learned indexes usually have greater p99 latency than the B+-tree for Lookup-Only and Write-Only workloads, and exhibit less stable performance.

Evaluation – Q5: Impact of Buffer



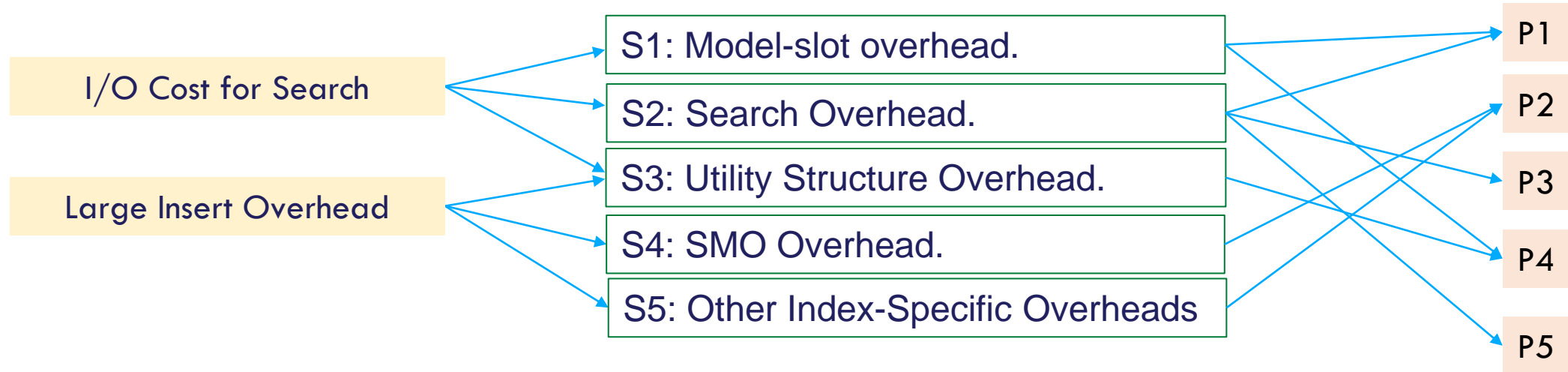
- ✓ With zero or only a small buffer size, LIPP has the smallest fetched.
- ✓ A larger buffer size can boost other indexes significantly.

Summary of Evaluation



- B+-tree is a most-of-the-time winner in both workload performance and storage cost!
- B+ tree shows more stable performance across different data distributions and workloads, while learned indexes fluctuate.

Our Guided Design Principles



P1: Reducing the Tree Height of the Index.

P2: Light-weight structural modification operation.

P3: Lower overhead in fetching next item.

P4: Storage Layout design

P5: Co-design learned index when using buffer.

More...

- Experiments on other datasets
- Experiments on SSD, inner nodes in main memory, ...

Conclusion

- Revisited and implemented representative learned indexes
- Conducted a comprehensive experiment study across various workloads and settings.
- Identified the shortcomings and proposes a guideline for design choice towards building a better learned index on disk



<https://github.com/rmitbgroup/LearnedIndexDiskExp>

Thanks!