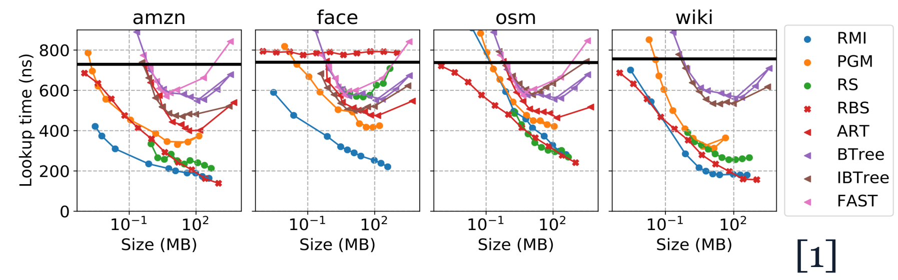
Tuning Hierarchical Learned Indexes on Disk and Beyond

Supawit Chockchowwat
Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign

Motivation

Learned indexes outperform in internal memory setting

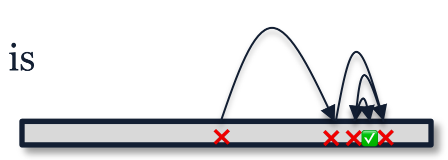


[1]

Large data systems require external storages

Round-trip time violates fast-random-access assumption

e.g., error correction is slower



Are fast random accesses necessary?

How to design or tune a learned index on external storage?

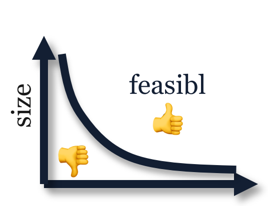
Challenges

Storages have their unique performance profiles.

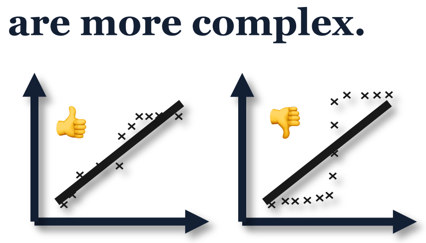
Index structures gain more design choices.

e.g., model, loss function, error distribution

Objective and constraints are more complex.



Pareto size and accuracy

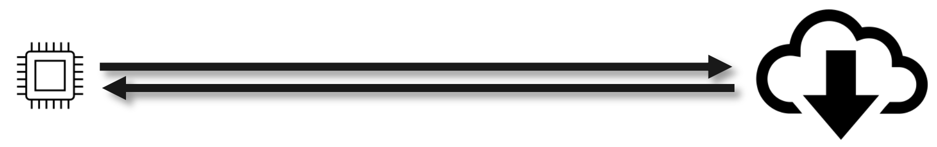


Model preference

ILLINOIS

Objective: Cost under Ext. Mem. Model

Cost to access external memory dominates the total cost.



Represent the time to read x bytes from storage as $T(x)$

e.g., affine storage profile: $T(x) = \frac{x}{bandwidth} + latency$

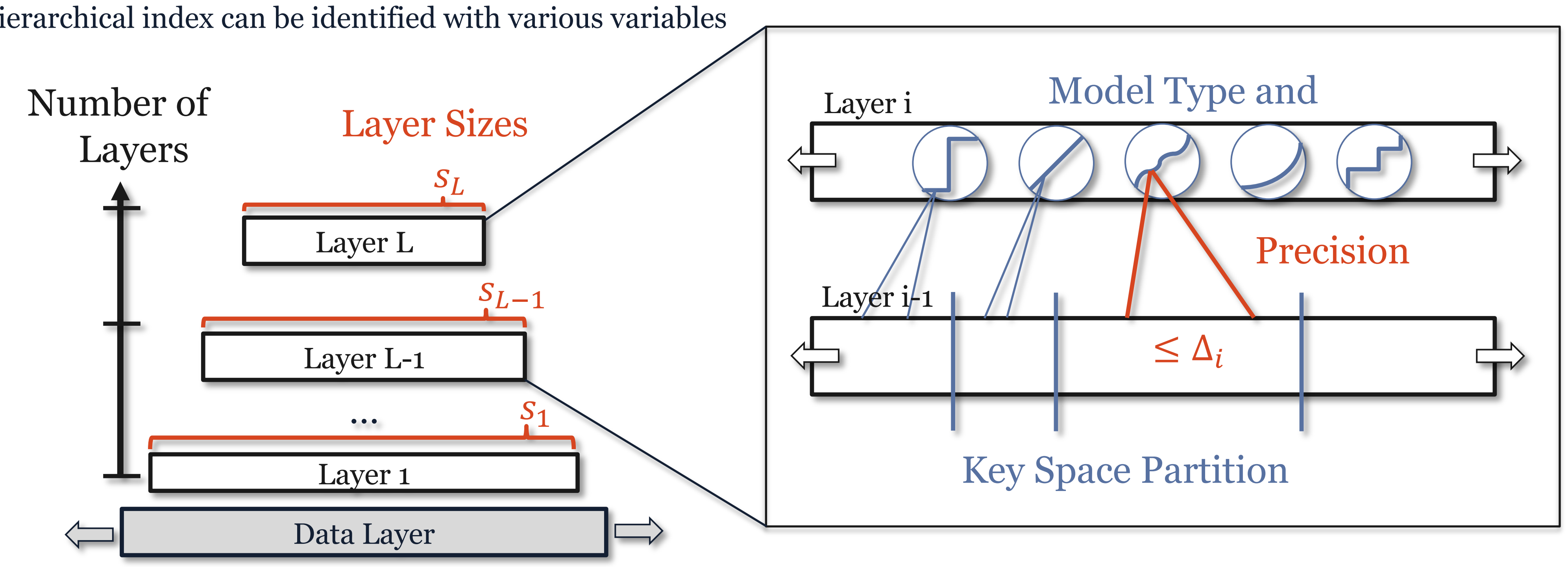
Index search cost under EMM

$T(s_L)$	retrieve root layer
$+\,T(\Delta_L)$	retrieve layer L-1
$+\,T(\Delta_{L-1})$	retrieve layer L-2
$+\,\ldots$	
$+\,T(\Delta_1)$	retrieve data layer

s_i: i-th layer size, Δ_i: precision at i-th layer

Search Space: Hierarchical Index

Hierarchical index can be identified with various variables

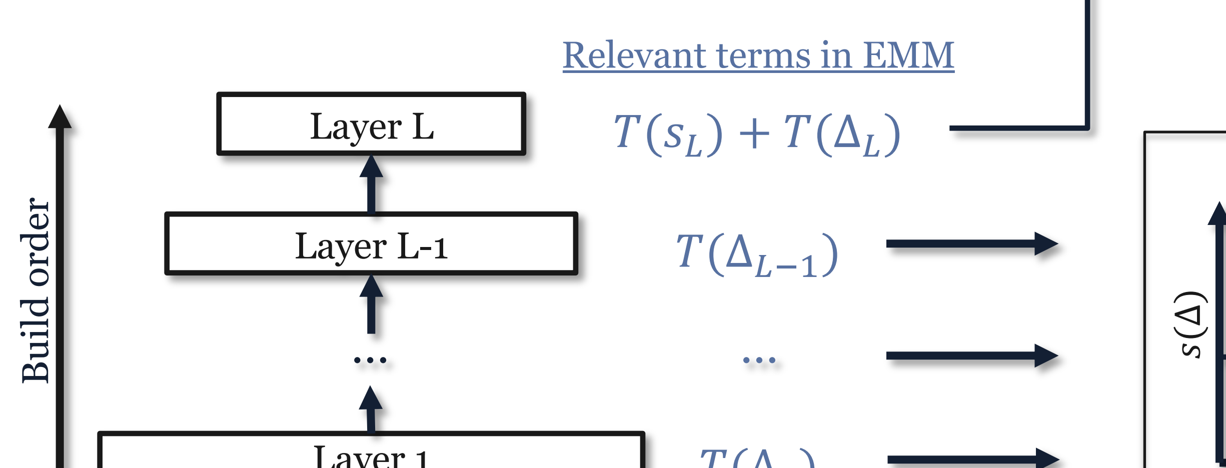


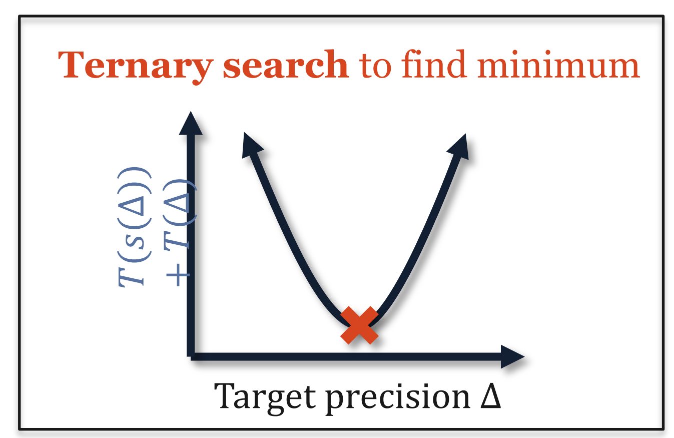
These variables are dependent on each other.

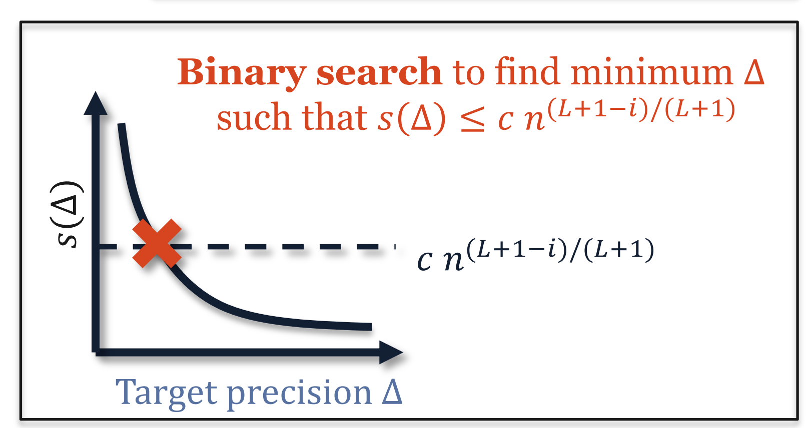
Optimization: Greedy Stack and Balance

Incrementally try number of layers to solve $L^* = \mathrm{argmin}_{L\geq 0}\; T(s_L) + \sum_{i=1}^{L} T(\Delta_i)$

For each number of layer L, build from bottom up with bounded size.







Given a precision Δ and a model type, AirIndex greedily partitions the key space and generates a layer of size s.

e.g., greedy packing for step functions, convex hull packing for piecewise linear function

Experiments

Baselines: PostgreSQL [2], RocksDB [3], RMI [4]

Benchmark: Search On Sorted Data (SOSD) [1, 5]

1. **books**: Amazon sale popularity data
2. **fb**: Facebook user IDs, upsampled
3. **osm**: OpenStreetMap locations in Google S2 Cellids
4. **wiki**: Wikipedia article edit timestamps

Environment: Azure VM D4s_v3 (4 vCPUs, 16 GiB RAM), data and indexes on Azure NFS

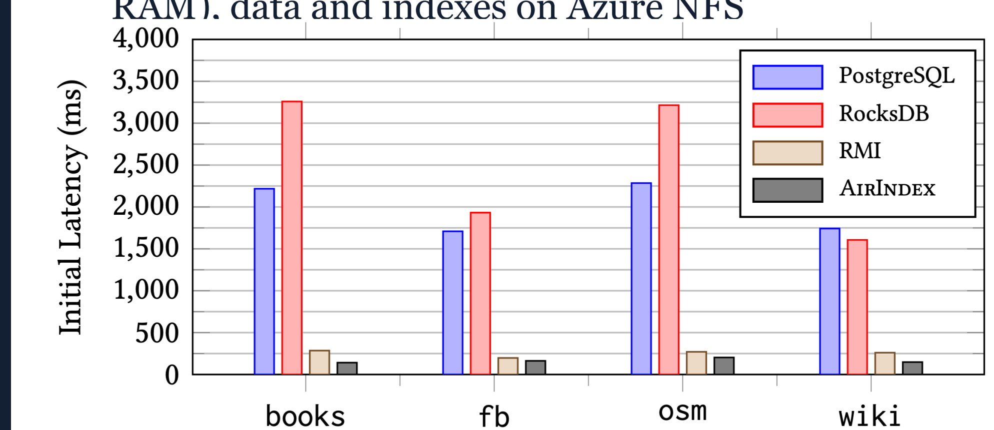


Figure 1: Initial lookup latency across SOSD datasets of different systems whose external memory is on Azure NFS.

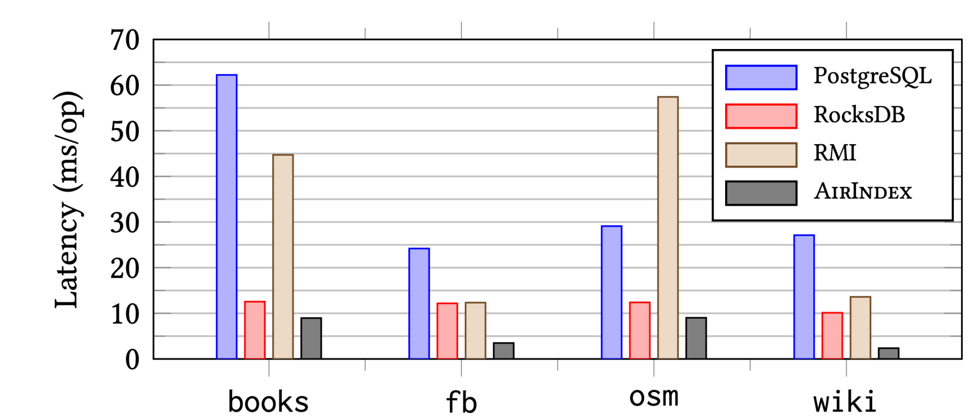


Figure 2: Average Lookup latency (over 70k queries) across SOSD datasets of different systems on Azure NFS.

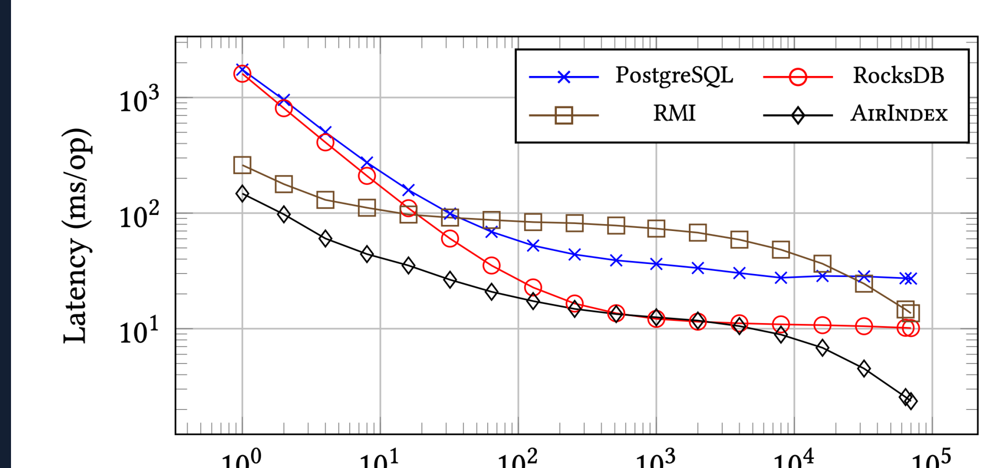


Figure 3: Average latency over first n queries of different systems on wiki dataset. The plot shows $n \in \{1, 2, 4, \ldots, 512, 1k, 2k, 4k, \ldots, 64k, 70k\}$. Both axes are in the logarithmic scale.

References

[1] Ryan Marcus, Andreas Kipf, Alexandervan Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking Learned Indexes. Proc. VLDB Endow. 14, 1 (2020), 1–13.

[2] PostgreSQL. [n. d.]. PostgreSQL: The World's Most Advanced Open Source Relational Database. https://www.postgresql.org. [Online; accessed November- 12-2021].[3] RocksDB

[3] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-Value Store Serving Large-Scale Applications. ACM Trans. Storage 17, 4, Article 26 (Oct. 2021), 32 pages. https://doi.org/10.1145/3483840

[4] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. Proceedings of the ACM SIGMOD International Conference on Management of Data. https://doi.org/10.1145/3183713.3196909

[5] Andreas Kipf, Ryan Marcus, Alexandervan Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2019. SOSD: A Benchmark for Learned Indexes. NeurIPS Workshop on Machine Learning for Systems (2019)