Learned Indexes

CS 6530, Fall 2023

Hello

- I’m Yuvaraj Chesetti
  - Yuvi (as in You-Vee or UV) or Yuva (as in You Va) works!
- PhD Student since Summer 2023
  - Joined as a Masters Student in Fall 2022, converted
- Learning/Researching about Learned Indexes
  - Started out in the CS 6530 Fall project
- Catch me in the lab!
  - Always happy to discuss

Overview

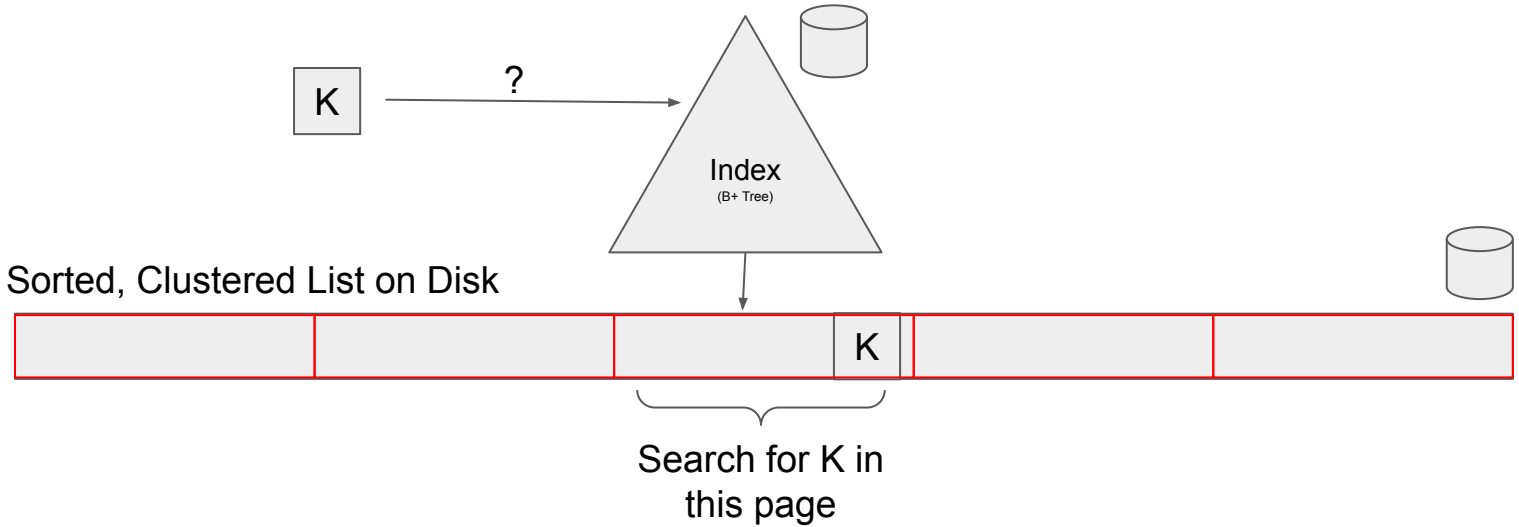
- Main Idea
- Details
  - Read only learned index
  - Updateable learned index
- Research
  - Published
  - Ongoing

Indexes

- Given a key K in a sorted list, where is it located?
  - Look up an associated value with at that location
  - Unsorted List? Store a sorted list of pointers and treat the same way
  - Some indexes only check for indexes (Hash Table)
- Examples
  - Content section of a book (primary index, primary key is chapters)
  - Word Index (secondary index, keyed on words)
- If Data is sorted, why build an index?
  - Linear Search - O(n/B) fetches, (works for unsorted data)
  - Binary Search - O(log(n)) fetches, no locality
  - Fetch (In-Memory, Cache Misses, External-Memory - Disk IO Blocks)

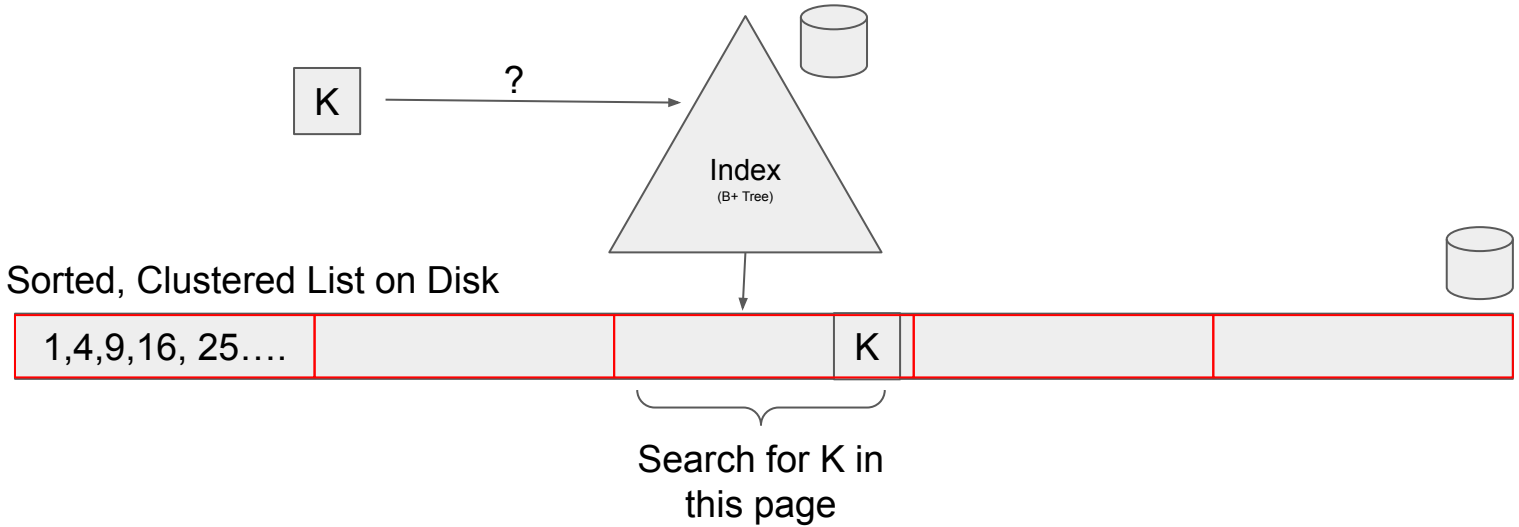
B+ Trees

- Spend extra memory organizing an index, you could speed up queries!
- B+ Tree
  - $O(\log_B(n))$ fetches, Takes advantage of block based IOs in external memory
  - Go to default for building indexes



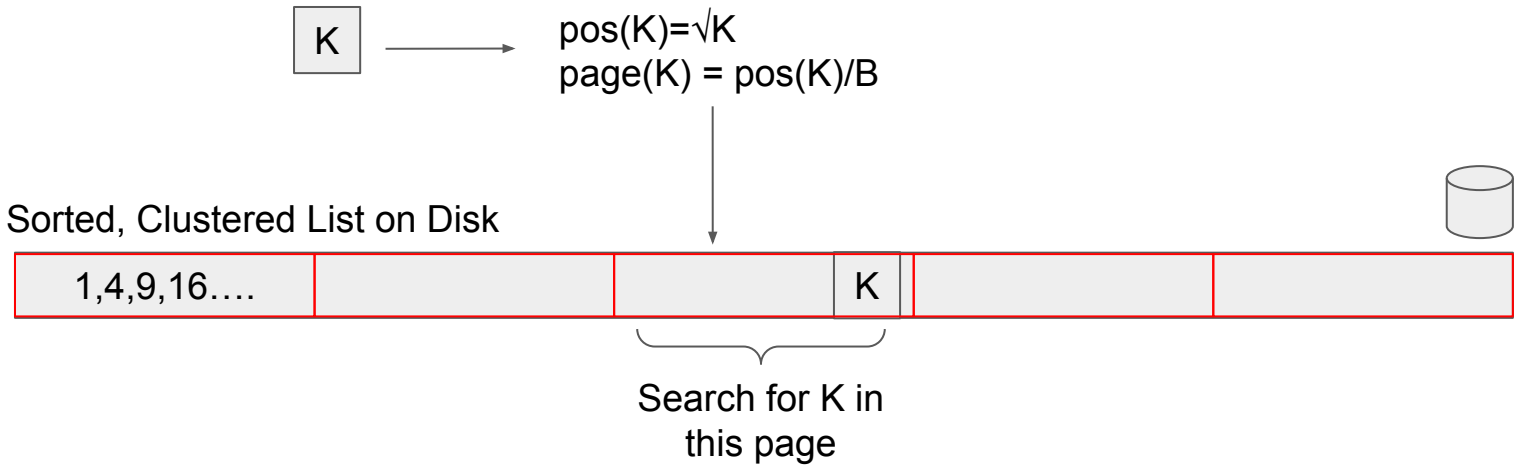
B+ Trees - One size fits all

- If data followed a pattern, do you need the B+ Tree?



Model the data

- No! - Just model the function



Kraska, Tim, et al. "The case for learned index structures." SIGMOD 2018.

The Case for Learned Index Structures

Tim Kraska* MIT kraska@mit.edu
Alex Beutel Google, Inc. abeutel@google.com
Ed H. Chi Google, Inc. edchi@google.com
Jeffrey Dean Google, Inc. jeff@google.com
Neoklis Polyzotis Google, Inc. npoly@google.com

ABSTRACT

Indexes are models: a B-Tree-Index can be seen as a model to map a key to the position of a record within a sorted array, a Hash-Index as a model to map a key to a position of a record within an unsorted array, and a BitMap-Index as a model to indicate if a data record exists or not. In this exploratory research paper, we start from this premise and posit that all existing index structures can be replaced with other types of models, including deep-learning models, which we term *learned indexes*. We theoretically analyze under which conditions learned indexes outperform traditional index structures and describe the main challenges in designing learned index structures. Our

a set of continuous integer keys (e.g., the keys 1 to 100M), one would not use a conventional B-Tree index over the keys since the key itself can be used as an offset, making it an $O(1)$ rather than $O(\log n)$ operation to look-up any key or the beginning of a range of keys. Similarly, the index memory size would be reduced from $O(n)$ to $O(1)$. Maybe surprisingly, similar optimizations are possible for other data patterns. In other words, knowing the exact data distribution enables highly optimizing almost any index structure.

Of course, in most real-world use cases the data do not perfectly follow a known pattern and the engineering effort to build specialized solutions for every use case is usually too

Kraska, Tim, et al. "The case for learned index structures." SIGMOD 2018.

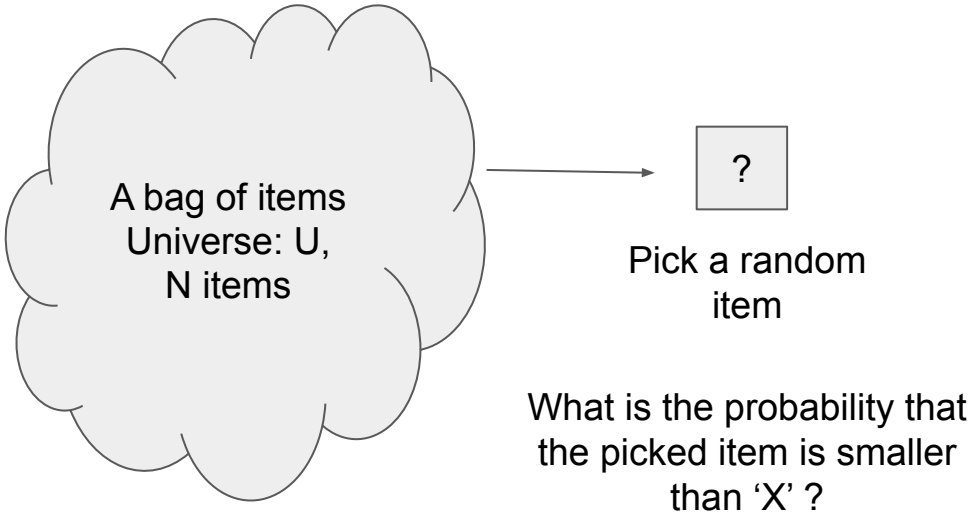
Classical data structures don’t take advantage of the distribution of data

- They are designed to work for all distributions in worst case
- Modelling is the bread and butter of ML techniques
  - Specific hardware is being designed for ML, can take advantage of that.
- Not just indexes - filters, hashmaps, joins, sorts, merges…

Learn?

- What is being learnt?
- How do you learn?
- What about error?

Cumulative Distributive Function

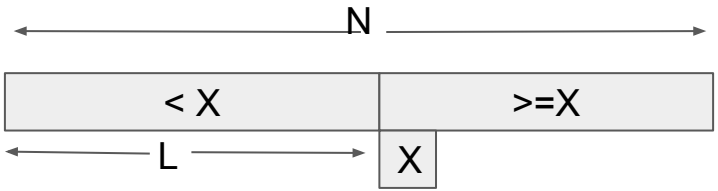


What is the probability that the picked item is smaller than ‘X’ ?

Cumulative Distributive Function

Pick a random item

What is the probability that the picked item is smaller than ‘X’ ?

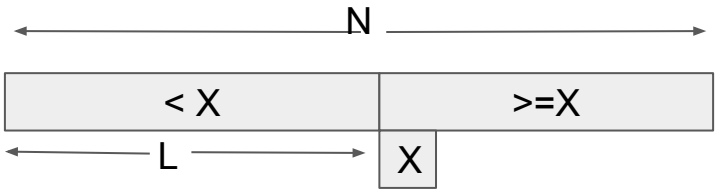


P(‘?’ < X) = L/N

Cumulative Distributive Function

Pick a random item

What is the probability that the picked item is smaller than ‘X’ ?



P(‘?’ < X) = L/N

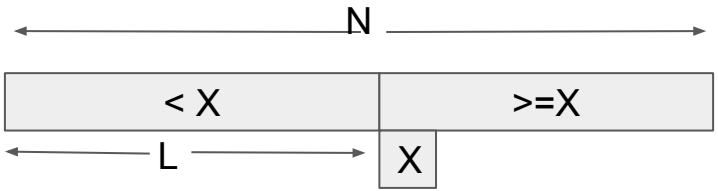
Pos(X) = P(‘?’ < X) * N

This is the position/lower bound of ‘X’ if the items in the bag were laid out sorted!

Cumulative Distributive Function

Pick a random item

What is the probability that the picked item is smaller than ‘X’ ?

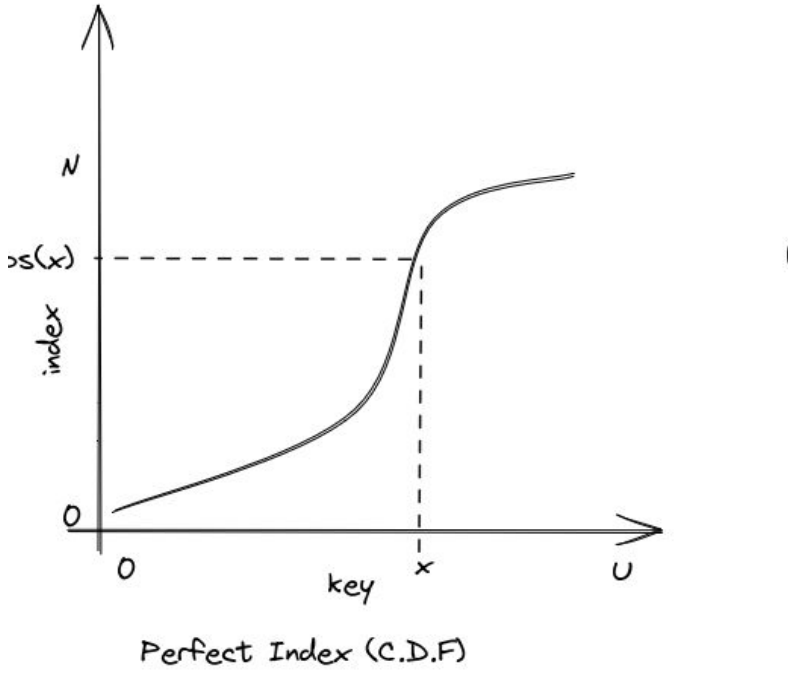


P(‘?’ < X) = L/N

Pos(X) = P(‘?’ < X) * N

This is the position/lower bound of ‘X’ if the items in the bag were laid out sorted!

Cumulative Distributive Function

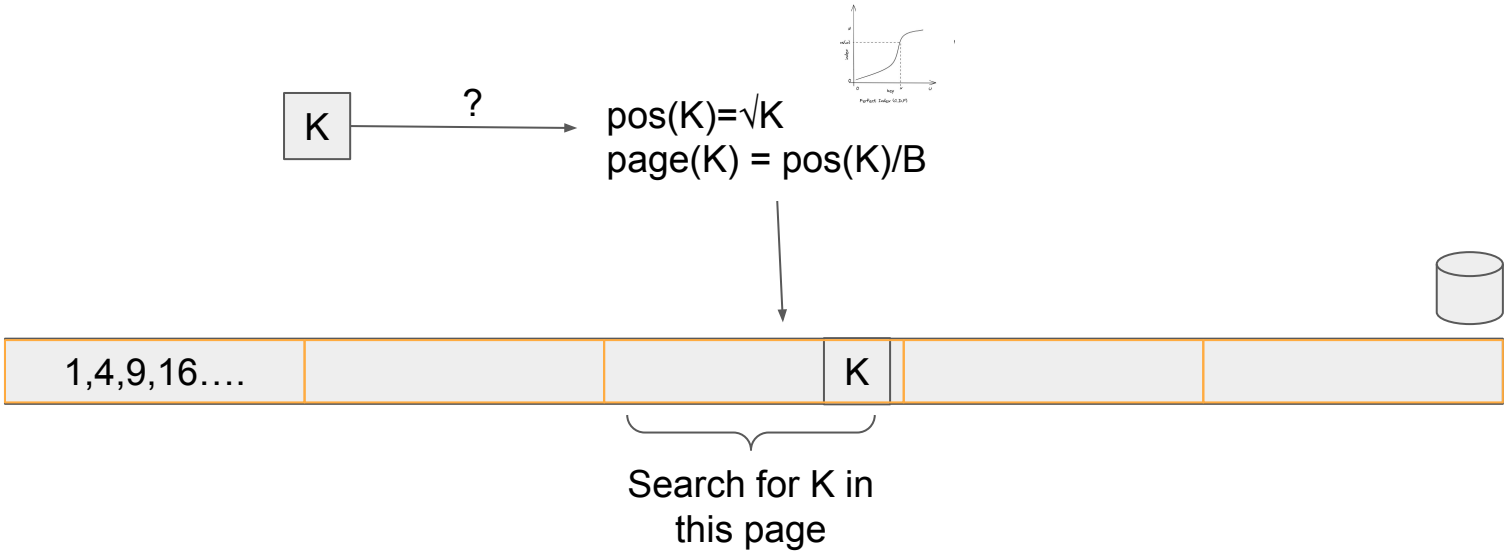


Perfect Index (C.D.F)

- P('?' < X) = L/N is called the Cumulative Distribution Function
- Integrating for x from (-∞, x) in the Probability Density Function
- Monotonic increasing function
- Learn the distribution = Learn the C.D.F
- All Indexes learn the C.D.F (Even B+ Trees)

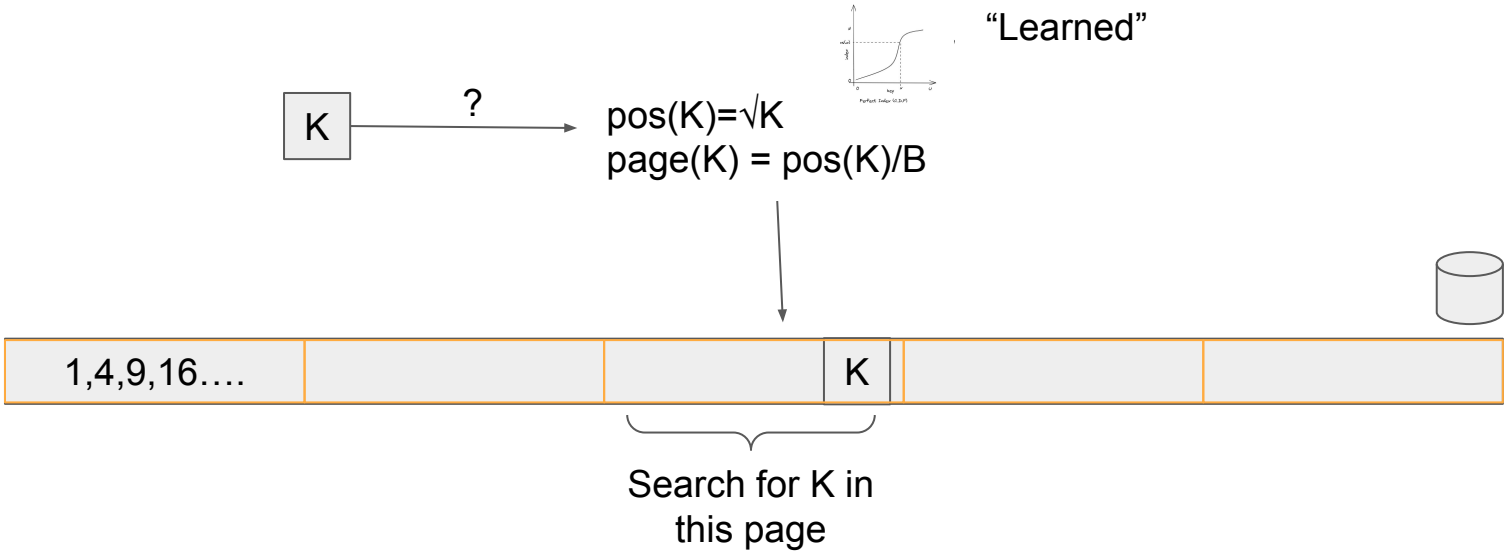
Revisiting the toy example

- CDF(X) = sqrt(K)/N
- Index(X) = (CDF(X) * N)



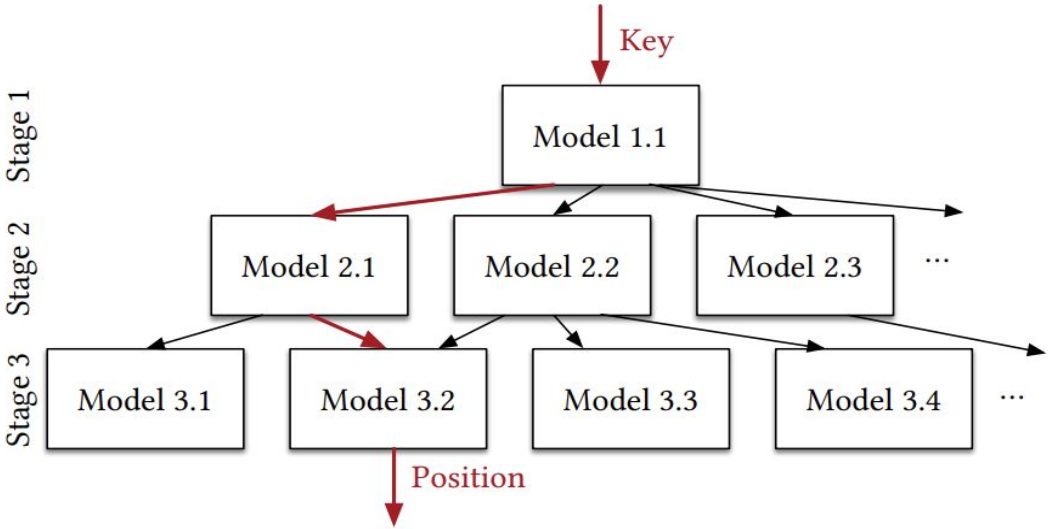
Revisiting the toy example

- CDF(X) = sqrt(K)/N
- Index(X) = (CDF(X) * N)



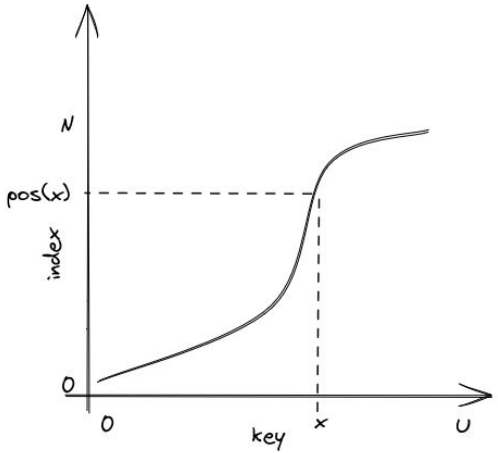
How do you model the CDF?

- Not all distributions are simple like our toy example
- Key Idea: Break it into smaller problems - the Recursive Model Index

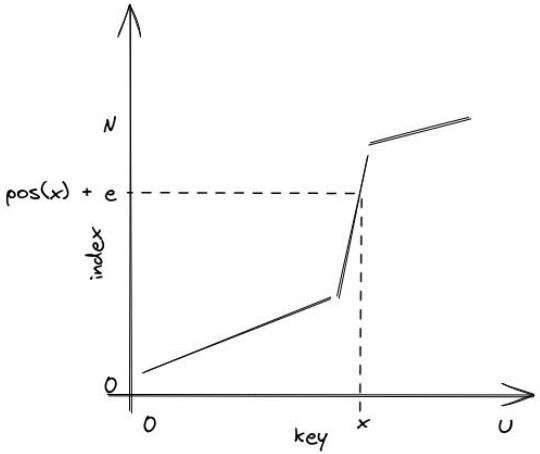


What about Individual Models?

- Neural Networks
  - Too heavy, designed for larger datasets
- RMI proposed needs to be tuned for levels, models, error
- Only linear functions works well (Piecewise Linear Approximation)

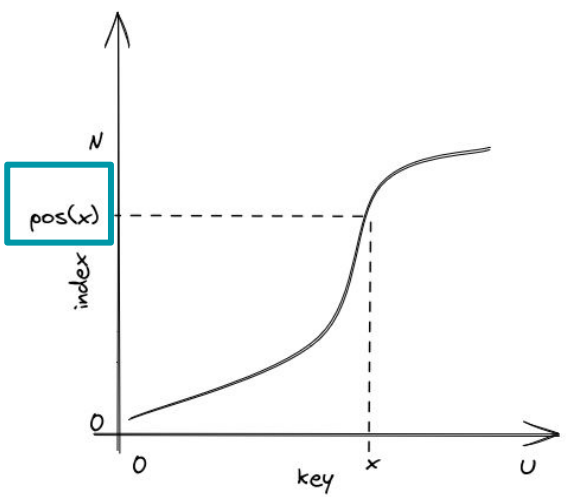


Perfect Index (C.D.F)

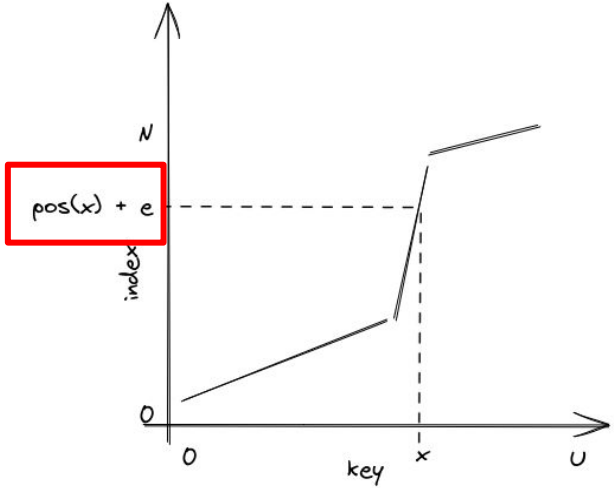


Learned Index (P.L.R of C.D.F.)

What about error?



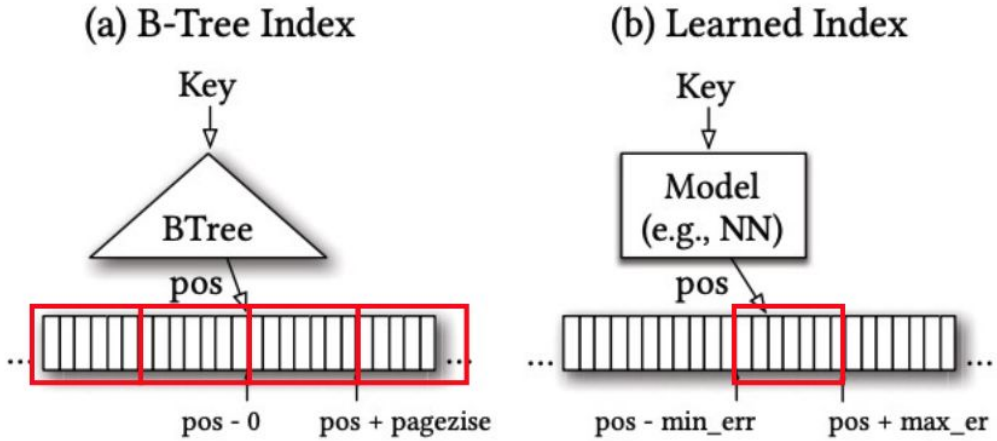
Perfect Index (C.D.F)



Learned Index (P.L.R of C.D.F.)

All Indexes are ‘Learned Indexes’ (even B+ Trees)

- All indexes have error!
- B-Tree has an error of page-size (overfit)
- Error can be bounded, Model Size inverse to Error



PGM Index - Inner Index complexity

Computational complexity

Let n be the number of keys, and B be the page size of the machine.

	PGM-index	B-tree	Self-balancing BST†	Skip list	Sorted array
Predecessor query§ (static case)	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(\log n)$
Predecessor query (dynamic case#)	$\mathcal{O}(\log_B^2 n)$	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(\log n)$
Insert/delete	$\mathcal{O}(\log_B n)$ amortised	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(n)$
Index space in words	$\mathcal{O}(\frac{n}{B^2})$ w.h.p.‡	$\mathcal{O}(\frac{n}{B})$	$\mathcal{O}(n)$	$\mathcal{O}(n)$ w.h.p.	$\mathcal{O}(1)$

Source: https://pgm.di.unipi.it/

PGM Index - Inner Index complexity

Computational complexity

Let n be the number of keys, and B be the page size of the machine.

	PGM-index	B-tree	Self-balancing BST[†]	Skip list	Sorted array
Predecessor query[§] (static case)	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(\log n)$
Predecessor query (dynamic case[#])	$\mathcal{O}(\log_B^2 n)$	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(\log n)$
Insert/delete	$\mathcal{O}(\log_B n)$ amortised	$\mathcal{O}(\log_B n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$ w.h.p.	$\mathcal{O}(n)$
Index space in words	$\mathcal{O}(\frac{n}{B^2})$ w.h.p.[‡]	$\mathcal{O}(\frac{n}{B})$	$\mathcal{O}(n)$	$\mathcal{O}(n)$ w.h.p.	$\mathcal{O}(1)$

(B = Error, inner query is faster, last mile is longer)

Source: https://pgm.di.unipi.it/

Search on Sorted Data Benchmark

Index / Index Size	XS Up to 0.01% of data size	S Up to 0.1% of data size	M Up to 1% of data size ▼	L Up to 10% of data size	XL No limit
RMI	374 ns	225 ns	**172 ns**	**151 ns**	**141 ns**
RS	**169 ns**	**156 ns**	203 ns	193 ns	184 ns
PGM	354 ns	303 ns	247 ns	228 ns	228 ns
ALEX		534 ns	430 ns	355 ns	298 ns
ART		562 ns	441 ns	396 ns	379 ns
BTree	806 ns	601 ns	524 ns	515 ns	514 ns
FAST		633 ns	544 ns	544 ns	544 ns
BinarySearch	680 ns	680 ns	680 ns	680 ns	680 ns

Learn?

- What are we trying to learn?
  - The Cumulative Distribution Function
- How do you learn?
  - Break it down into sub-pieces makes it easier
  - Different Learned Index implementations vary in exact implementation details
  - CDF approximation techniques - piecewise linear regression
- What about error?
  - It’s not a problem, it doesn’t hurt performance

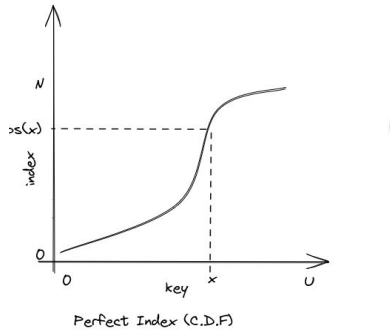
Learn?

- What are we trying to learn?
  - The Cumulative Distribution Function
- How do you learn?
  - Break it down into sub-pieces makes it easier
  - Different Learned Index implementations vary in exact implementation details
  - CDF approximation techniques - piecewise linear regression
- What about error?
  - It’s not a problem, it doesn’t hurt performance
- **What about updates?**

Updateability of Learned Indexes

So far, we've talked about sorted clustered data

- Where do you insert?
- When do you retrain?



Perfect Index (C.D.F)

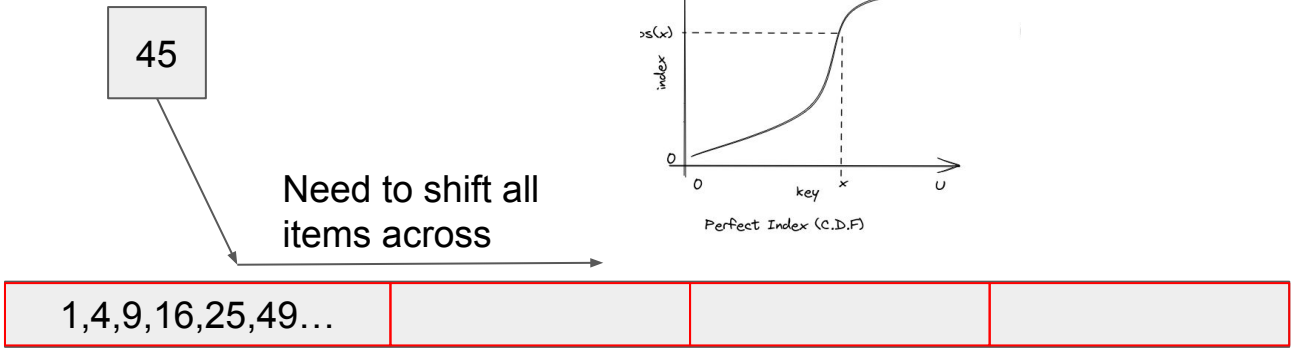
1,4,9,16,25,49…			

Sorted, Clustered List on Disk

Updateability of Learned Indexes

So far, we've talked about sorted clustered data

- **Where do you insert?**
- When do you retrain?

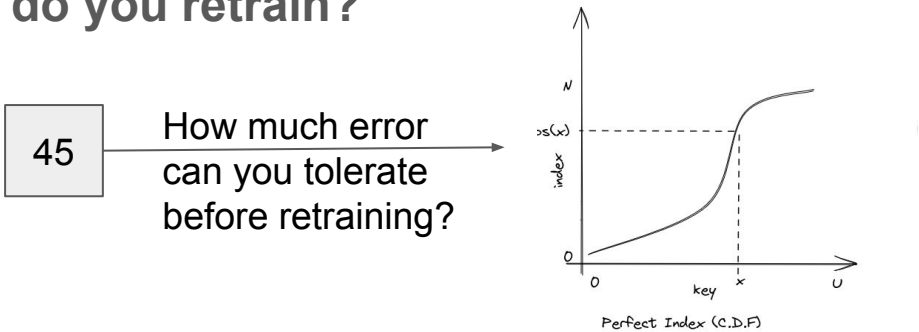


Sorted, Clustered List on Disk

Updateability of Learned Indexes

So far, we've talked about sorted clustered data

- Where do you insert?
- **When do you retrain?**



1,4,9,16,25,49…			

Sorted, Clustered List on Disk

B+ Tree is also a learned index

- B+ Tree is also a learned index - that handles updates just fine
- Can we combine ideas from the B+ Tree with a learned index?
- Where do you insert?
  - Internal nodes have gaps in between
- When do you retrain?
  - The B+ Tree splits and merges nodes as the incoming distribution changes
- But a B+ Tree does not know anything about the the data distribution?
  - It doesn’t really know the distribution, it just adapts to the worst case
  - It is not an efficient ‘learning’ approach (Overfits)
  - I lied

ALEX: An Updatable Adaptive Learned Index

Jialin Ding[†] Umar Farooq Minhas[‡] Jia Yu[§]
Chi Wang[‡] Jaeyoung Do[‡] Yinan Li[‡] Hantian Zhang[∓] Badrish Chandramouli[‡]
Johannes Gehrke[‡] Donald Kossmann[‡] David Lomet[‡] Tim Kraska[†]
[†]Massachusetts Institute of Technology [‡]Microsoft Research [§]Arizona State University
[∓]Georgia Institute of Technology

ABSTRACT

Recent work on "learned indexes" has changed the way we look at the decades-old field of DBMS indexing. The key idea is that indexes can be thought of as "models" that predict the position of a key in a dataset. Indexes can, thus, be learned. The original work by Kraska et al. shows that a learned index beats a B+Tree by a factor of up to three in search time and by an order of magnitude in memory footprint. However, it is limited to static, read-only workloads.

index for dynamic workloads that effectively combines the core insights from the Learned Index with proven storage & indexing techniques to deliver great performance in both time and space? Our answer is a new in-memory index structure called ALEX, a fully dynamic data structure that simultaneously provides efficient support for point lookups, short range queries, inserts, updates, deletes, and bulk loading. This mix of operations is commonplace in online transaction processing (OLTP) workloads [6, 8, 32] and is also supported by B+Trees [29].

Implementing writes with high performance requires a

My Takeaway: Combine a B+ Tree structure with learned indexes, result: ALEX

ALEX: An Updatable Adaptive Learned Index

Jialin Ding[†] Umar Farooq Minhas[‡] Jia Yu[§]
Chi Wang[‡] Jaeyoung Do[‡] Yinan Li[‡] Hantian Zhang[∓] Badrish Chandramouli[‡]
Johannes Gehrke[‡] Donald Kossmann[‡] David Lomet[‡] Tim Kraska[†]
[†]Massachusetts Institute of Technology [‡]Microsoft Research [§]Arizona State University
[∓]Georgia Institute of Technology

ABSTRACT

Recent work on "learned indexes" has changed the way we look at the decades-old field of DBMS indexing. The key idea is that indexes can be thought of as "models" that predict the position of a key in a dataset. Indexes can, thus, be learned. The original work by Kraska et al. shows that a learned index beats a B+Tree by a factor of up to three in search time and by an order of magnitude in memory footprint. However, it is limited to static, read-only workloads.

index for dynamic workloads that effectively combines the core insights from the Learned Index with proven storage & indexing techniques to deliver great performance in both time and space? Our answer is a new in-memory index structure called ALEX, a fully dynamic data structure that simultaneously provides efficient support for point lookups, short range queries, inserts, updates, deletes, and bulk loading. This mix of operations is commonplace in online transaction processing (OLTP) workloads [6, 8, 32] and is also supported by B+Trees [29].

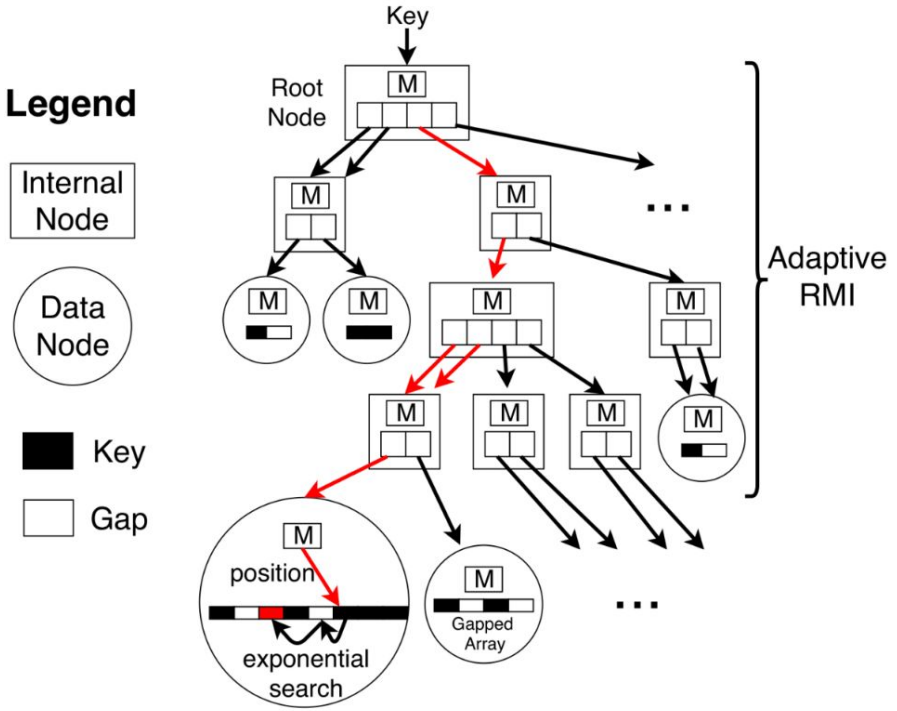
Implementing writes with high performance requires a

ALEX design overview

Structure
- **Dynamic tree** structure
- Each **node** contains a **linear model**
  - **internal** nodes → **models select the child** node
  - **data** nodes → **models predict the position** of a key

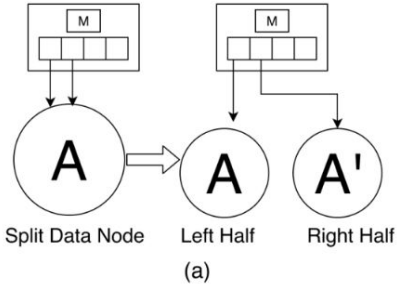
Core operations
- Lookup
  - Use **RMI to predict location of key** in a data node
  - Do **local search to correct** for prediction **error**
- Insert
  - Do a **lookup to find the insert** position
  - **Insert the new key/value** (might require shifting)

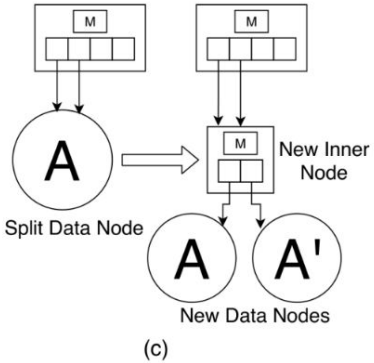
Current design constraints
a) In memory
b) Numeric data types
c) Single threaded

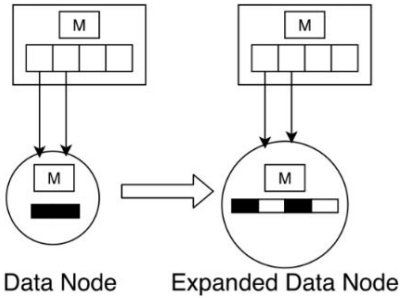


4. Adaptive Structure

- Flexible tree structure
  - Split nodes sideways
  - Split nodes downwards
  - Expand nodes
  - Merge nodes, contract nodes
- Key idea: all decisions are made to maximize performance
  - Use cost model of query runtime
  - No hand-tuning
  - Robust to data and workload shifts







ALEX - Few other optimizations

- Gapped Array
- Model Based Insertion
- Exponential Search

Gapped Array

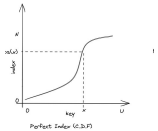
B+ Tree node
Dense, sorted

Gapped Array
Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

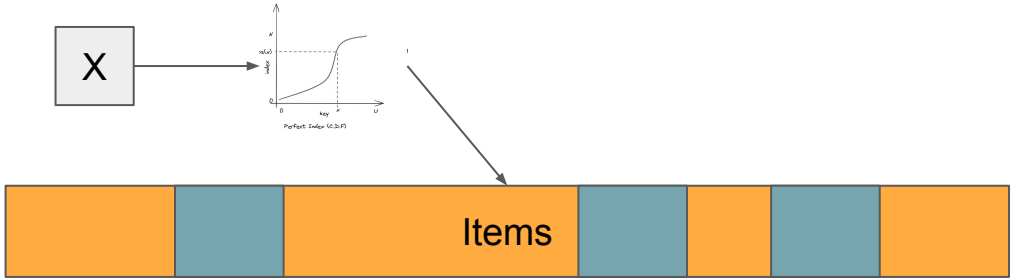
X



Gapped Array
Inserts more efficient
(less items to shift)

Model Based Insertion

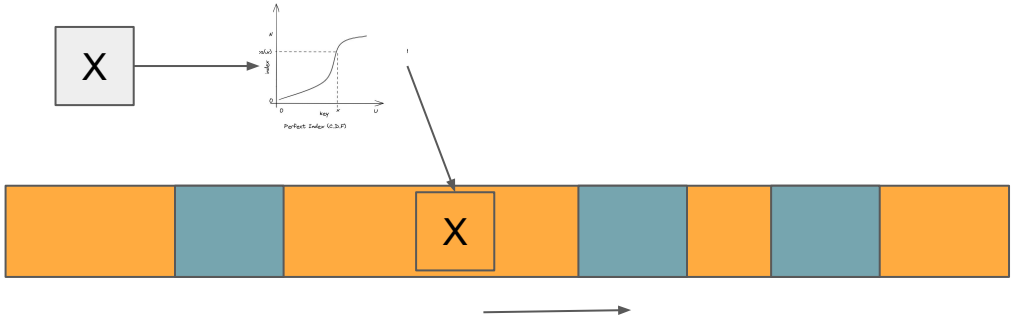
- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur



Gapped Array
Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

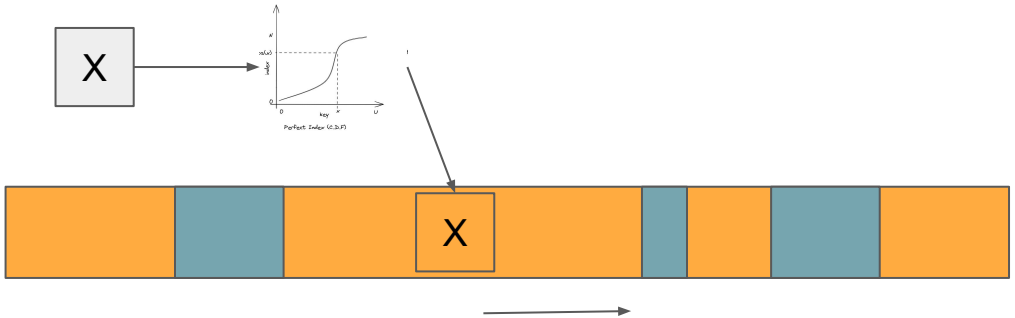


Gapped Array
Inserts more efficient
(less items to shift)

Shift other items to
nearest gap

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

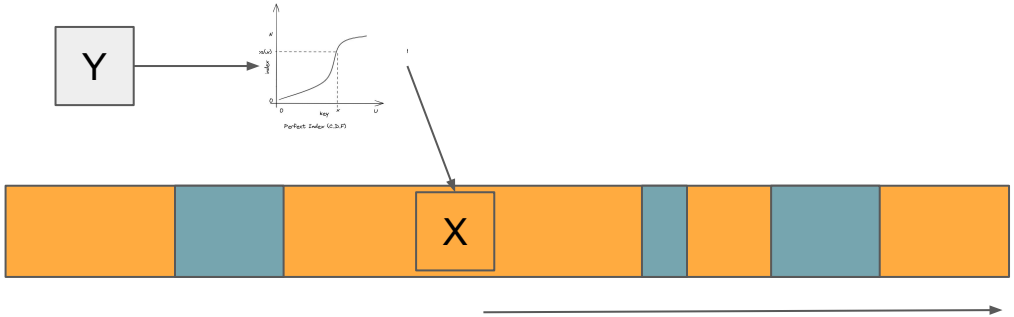


Gapped Array
Inserts more efficient
(less items to shift)

Shift other items to
nearest gap

Lookups in ALEX

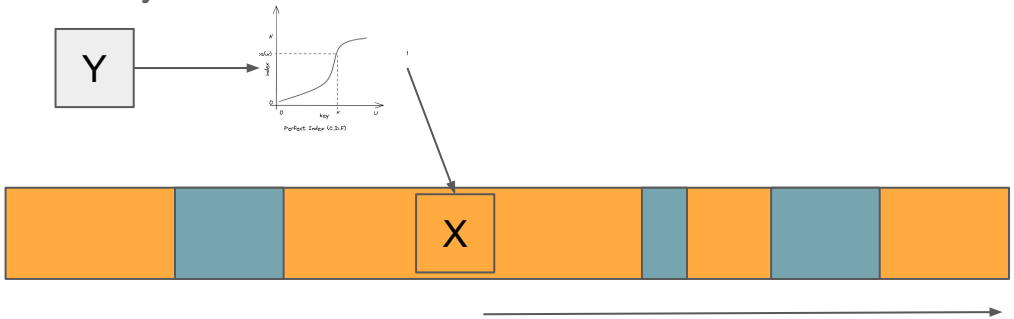
- Use the RMI to reach the correct Gapped Array
- Model based insertion - items will always be at or right of predicted position
- Start search from predicted position



Start search for Y from model predicted position

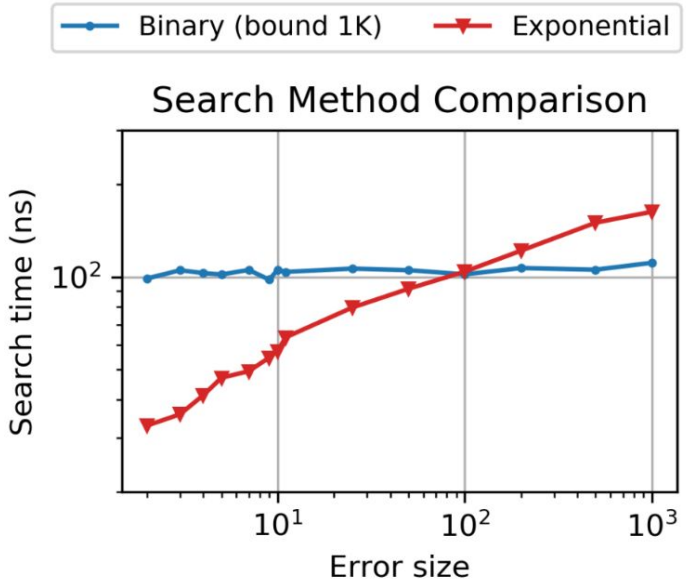
Search Algorithm

- Search - linear, binary or exponential?
- Exponential -
  - Search in windows of 2, 4, 8, 16… 2^x
  - If you overshoot, search in the second half with same window



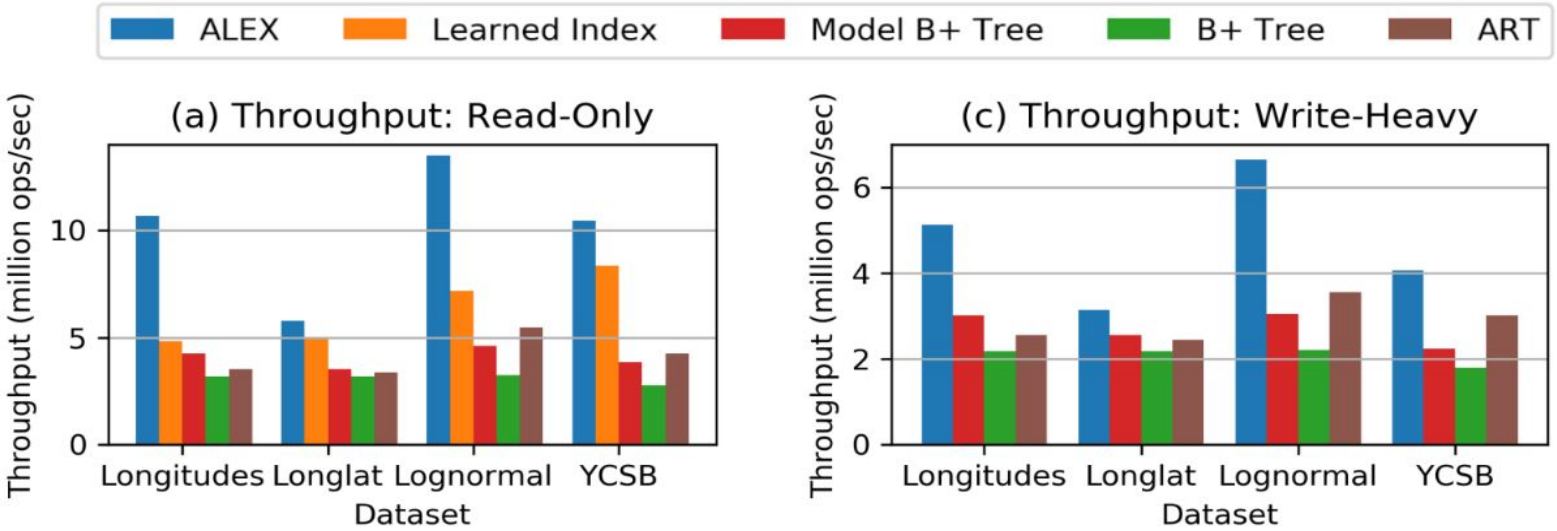
Start search for Y from model predicted position

3. Exponential Search



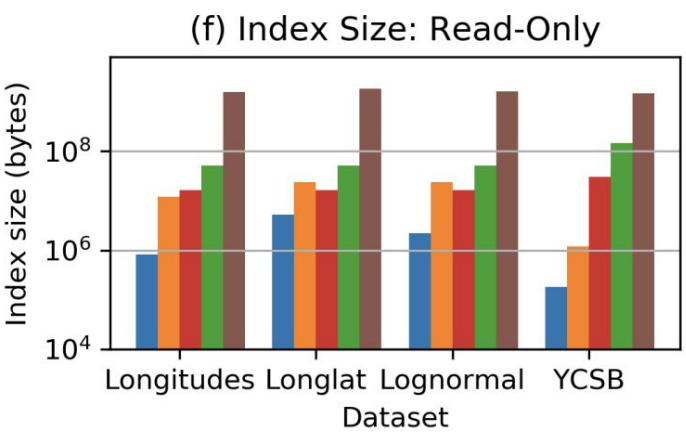
Model errors are low, so exponential search is faster than binary search

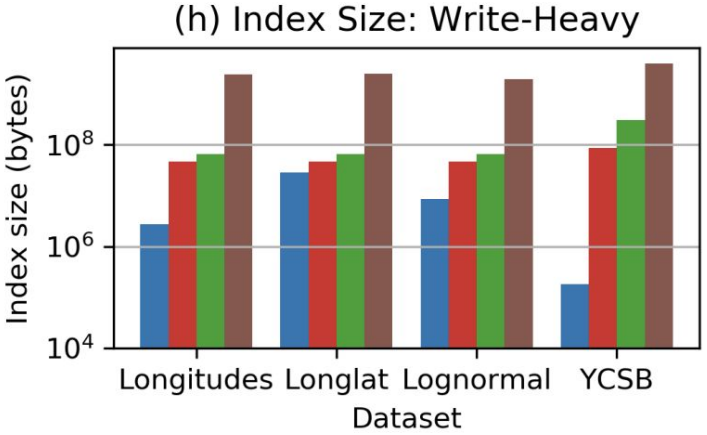
Slide from CS 6530, Fall 2022



~4x faster than B+ Tree
~2x faster than Learned Index

~2-3x faster than B+ Tree





~3 orders of magnitude less space for index

Different Learned Indexes

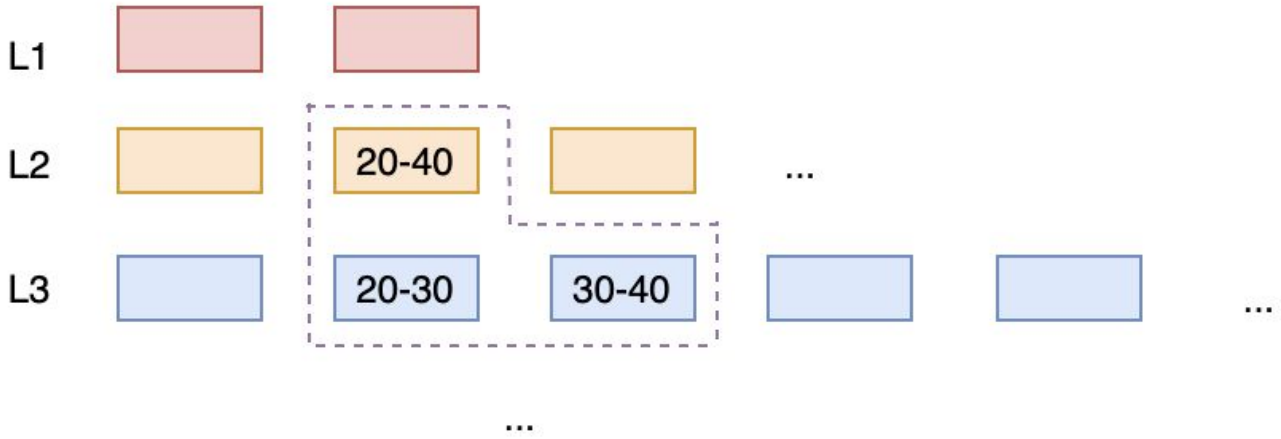
- RMI
- PGM Index - Ferragina, Paolo, and Giorgio Vinciguerra. "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds." *Proceedings of the VLDB Endowment* 13.8 (2020): 1162-1175.
- ALEX - Ding, Jialin, et al. "ALEX: an updatable adaptive learned index." *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020
- FITing Tree - Galakatos, Alex, et al. "Fiting-tree: A data-aware index structure." *Proceedings of the 2019 international conference on management of data*. 2019.
- LIPP - Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, Chunxiao Xing: Updatable Learned Index with Precise Positions. Proc. VLDB Endow. 14(8): 1276-1288 (2021).

Learned Index Performance

- Benchmark
  - SOSD - Marcus, Ryan, et al. "**Benchmarking learned indexes**.", NeurIPS Workshop on Machine Learning for Systems
  - Lan, Hai, et al. "**Updatable Learned Indexes Meet Disk-Resident DBMS-From Evaluations to Design Choices**." *Proceedings of the ACM on Management of Data* 1.2 (2023): 1-22.
- Theoretical
  - Ferragina, Paolo, Fabrizio Lillo, and Giorgio Vinciguerra. "**Why are learned indexes so effective?**." *International Conference on Machine Learning*. PMLR, 2020.
  - Sabek, Ibrahim, et al. "**Can Learned Models Replace Hash Functions?.**" *Proceedings of the VLDB Endowment* 16.3 (2022): 532-545.

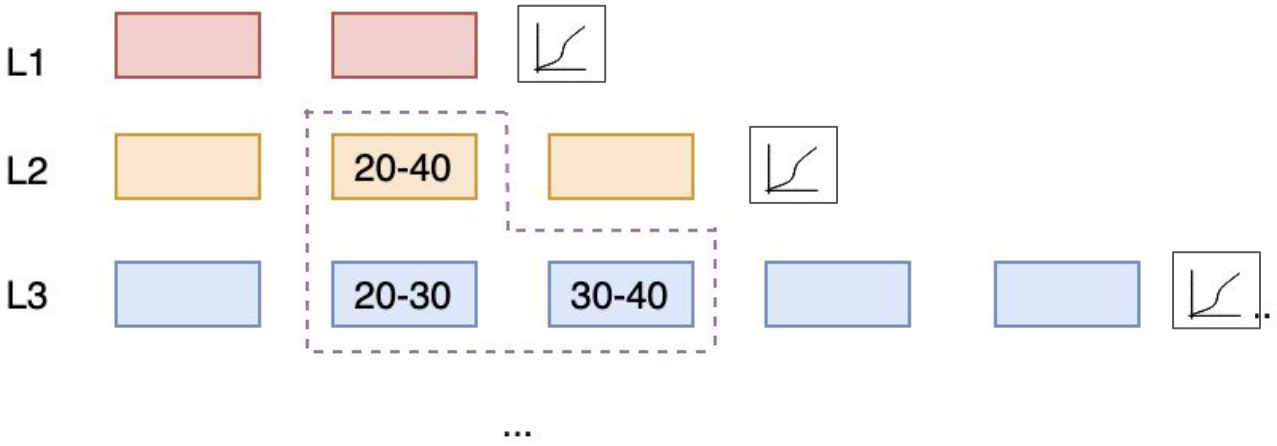
LSM Trees

- Dai, Yifan, et al. "**From {WiscKey} to Bourbon: A Learned Index for {Log-Structured} Merge Trees**." *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 2020.
- Abu-Libdeh, Hussam, et al. "**Learned indexes for a google-scale disk-based database**." *arXiv preprint arXiv:2012.12501* (2020).



LSM Trees

- Dai, Yifan, et al. "**From {WiscKey} to Bourbon: A Learned Index for {Log-Structured} Merge Trees**." *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 2020.
- Abu-Libdeh, Hussam, et al. "**Learned indexes for a google-scale disk-based database**." *arXiv preprint arXiv:2012.12501* (2020).



Other

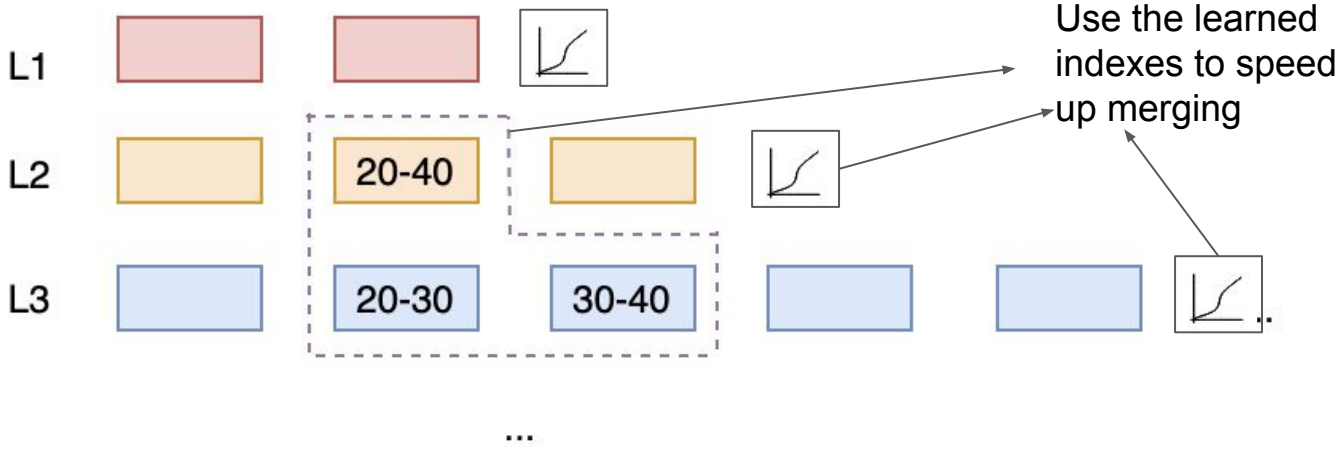
- Genomics
  - Ho, Darryl, et al. "**Lisa: Learned indexes for DNA sequence analysis**." *bioRxiv* (2020).
  - Kirsche, Melanie, Arun Das, and Michael C. Schatz. "**Sapling: Accelerating suffix array queries with learned data models**." *Bioinformatics* 37.6 (2021): 744-749.
- Spatial Indexing
  - Varun Pandey, Alexander van Renen, Andreas Kipf, Jialin Ding, Ibrahim Sabek, Alfons Kemper: **The Case for Learned Spatial Indexes.** AIDB@VLDB 2020
- Classical Algorithms
  - Kristo, Ani, et al. "**The case for a learned sorting algorithm**." *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020.
  - Sabek, Ibrahim, and Tim Kraska. "**The Case for Learned In-Memory Joins**." *arXiv preprint arXiv:2111.08824* (2021). (VLDB 2023)

Open Research Problems

- Variable length keys
- String keys
- Compression
- Concurrency
- Updates
- Optimizations
- Theoretical lower bounds
- Specialized Hardware

Ongoing Research

- Speed up merges in LSM Trees
- Synthetic benchmarks
- Tried out in Bourbon (LevelDB fork with learned indexes)
  - Makes it worse!
  - Still think there's some implementation gap or compaction policy we need to explore
  -



Ongoing Research

- Disk Based Joins - Indexed Nested Loop Join
- Use Learned Index instead of B+ Tree

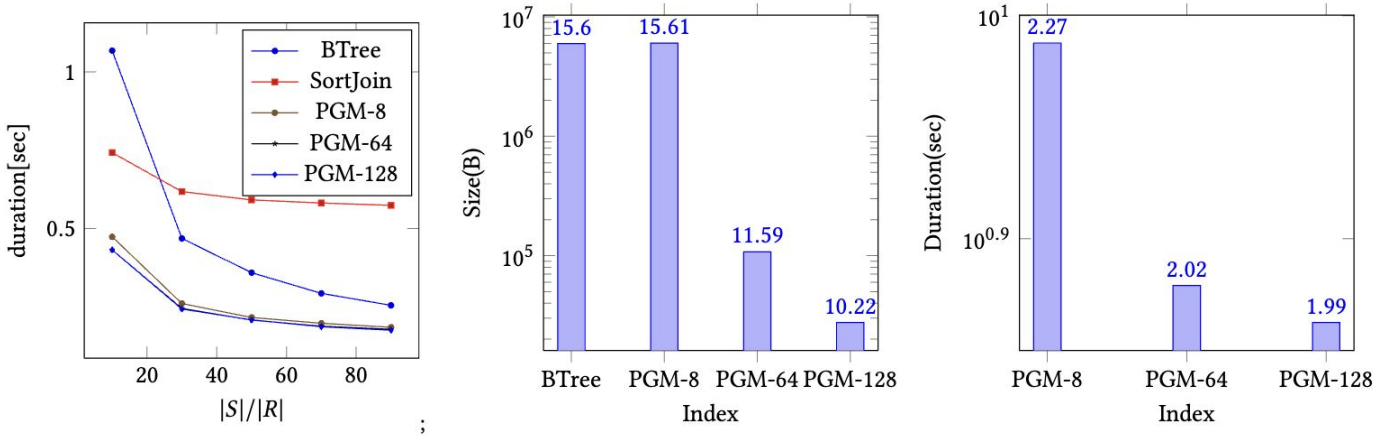


Figure 3: Dataset: Uniform Random $|S| = 200M$, 8byte keys, 8byte values, 4 Threads

Thanks!

Questions?