Alex : An updatable adaptive learned index

→ The key idea is that indexes can be thought of as "models" that predict the position of a key in a dataset.

→ a learned index beats B+ tree by a factor up to 3 in search time and by an order of magnitude in memory footprint.

However, it is limited to static - read-only workloads

⇒ A new learned index ALEX is presented;

it addresses the issues that arise when implementing learned indexes for workloads that contain a mix of point lookups, short range queries, inserts, updates and deletes.

ALEX effectively combines the core insights from learned indexes with proven storage and indexing techniques to achieve high performance and low memory footprint.

Kraska et al. refer to as "Learned Index" it proposes to replace a standard database index with DL models

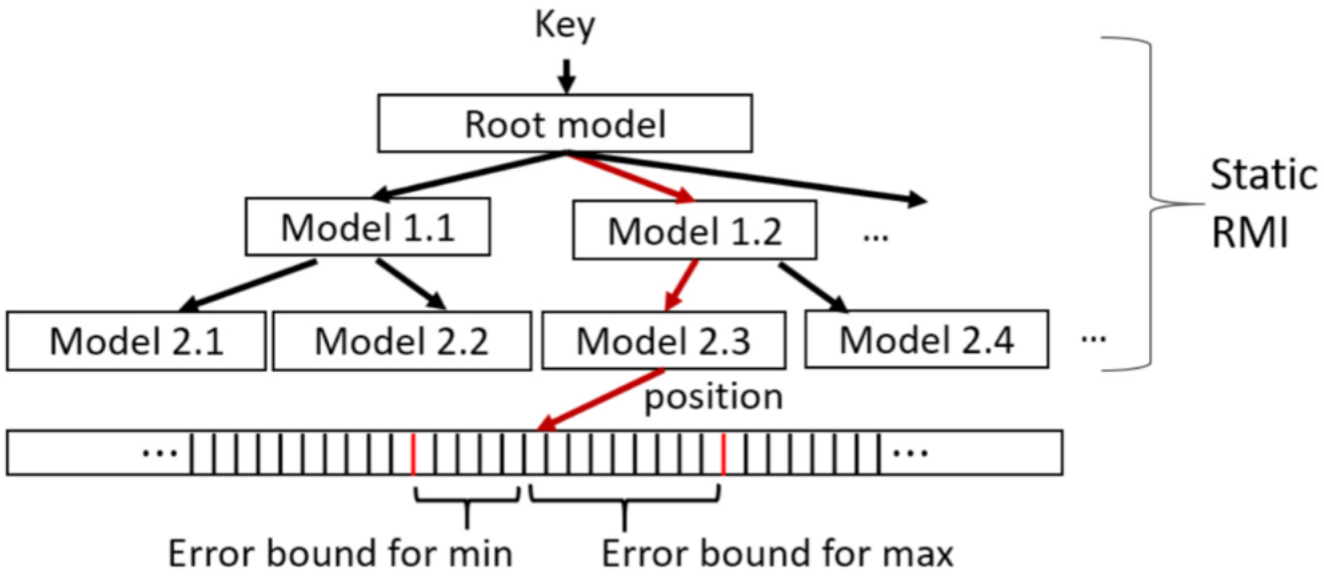

Figure 1: Learned Index by Kraska et al.

Given the key, an intermediate node in the hierarchy (Root Node) is used to predict the child model to use. (Model 1.1 and Model 1.2) and the leaf is a model to predict the location of the key in a densly packed array.

⇒ here they only use a simple models for "good enough" results. However this solution can handle only lookups

on read-only datasets which makes the learned index useless for dynamic read-write workloads, common in practice.

Implementing writes with high perf requires a careful design of the dataset (storing records).

Contributions:

- Storage layout optimized for models:
  it allows ~~#~~ nodes to grow and shrink at ≠ rates. To store records, ALEX uses an array with gaps (in a data node) (so it can amortize the cost of shifting the keys for each insertion because gaps can absorb inserts and also allows more accurate placement of data (ensure records are placed closely to the predicted position)
- Search strategy optimized for models:
  exploits model-based inserts combined with exponential search
- Keeping models accurate with dynamic data dist

and workloads. (it achieves this by exploiting adaptive expansion and node splitting mechanisms with selective model retraining.

- Detailed evaluation

⇒ On read-only datasets, ALEX beats the Learned Index by up to 2,2 x performance with up to 15 x smaller index size.

⇒ read-write workloads, Alex beats B+ tree by 4,1 x while never performing worse, with up to 2000 x smaller index size.

⇒ ALEX beats another enhanced B+ tree and the mem-opt Adaptive radix tree, scales to large data sizes and is robust to data distribution shift.

Background :

B^+ tree

The case of the learned Indexes: B+ tree indexes can be thought of as "models". If indexes are "models", techniques such as

they can be learned using trad[illegible]

CDF of the input data. ⇒ the resulting learned Index is optimized for the specific data distribution.

- Single NL model learned over the entire data is not accurate enough. To overcome this, ~~the~~ they used recursive model index (RMI)
  - RMI replaces the internal B+tree nodes with models
  - To support local search, keep min and max error bounds for each model in RMI and performs binary search within these bounds.
  - linear regression and NN models
  - The internal space is less than the B+tree's.

The main drawback:

→ it does not support any modifications.

Alex Overview:

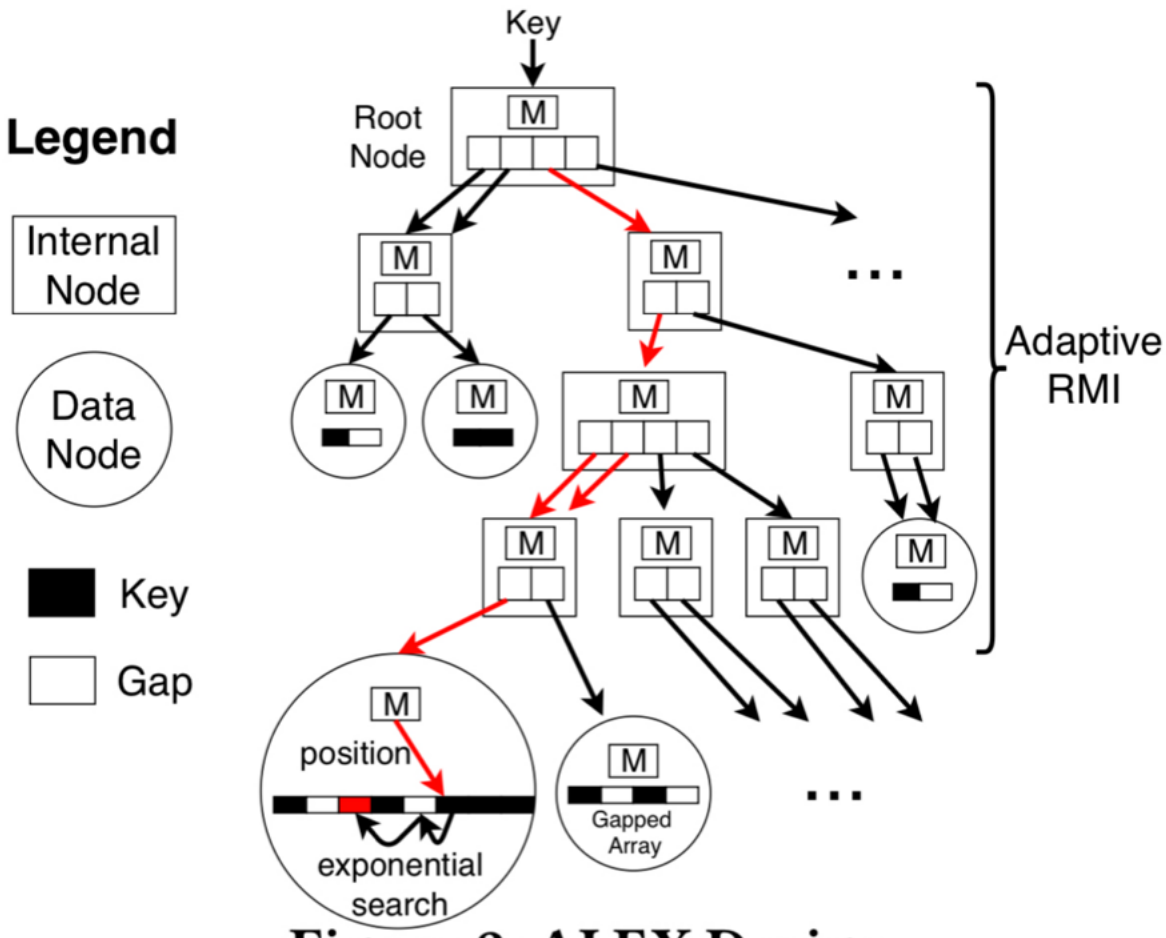

Figure 2: ALEX Design

→ Alex design takes adv of 2 key insights:

→ a space-time tradeoff leading to an adaptable data structure and faster lookups.

→ Alex used gapped Array layout for the leaf nodes

→ It uses dynamic RMI compared to the learned indexes that uses SRMI which performs poorly on inserts if the data dist is difficult to model.

→ Alex can be updated dynamically and efficiently at runtime and uses linear cost models that predict the latency of lookup and insert operations based on statistics measured from an RMI structure.

Alex goals:

→ Insert time should be competitive with B+ tree

→ lookup time should be faster than B+ tree and the learned Index

→ index storage space should be smaller.

→ Data storage space (leaf level) should be comparable to B+ tree.

Alex design Overview -

Alex an in memory updatable learned index.

Diff from "The learned index":

1. The data structure used to store the data at the leaf node. Alex uses a node per leaf and carefully leaves the free space
   - helps for faster inserts and lookup times.
   - (GA in case of Alex)
2. Uses exp search to find keys at the leaf level. to correct misprediction of the RMI.
   - exponential search is faster than binary search with boundaries (min-max) used "The learned Index"
3. Alex inserts keys into data nodes at the position where the models predicts where the key should be. "model-based insertion"
4. Alex dynamically adjusts the shape and height of the RMI.

5. It has no parameters that need to be re-tuned for each data set

Node layout

- Data nodes: like B^+ tree, the leaf nodes of Alex stores the data records and refered to as data nodes. Data node stores a linear regression model (2 double values for slope and intercept) and two gapped arrays (one for keys and one for payload)
  - Gapped array uses model-based inserts to distribute extra space between the elements of the array
- Internal Nodes: they store a LR model and an array containing pointers to children nodes. unlike B^+ tree, they use models to "compute" the location, in pointers array, of the next child pointer to follow.