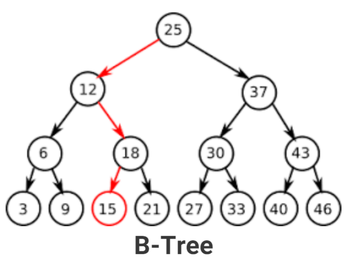
BenchLex: A Sorted Benchmark for ALEX

Clara Henzinger [1] Alexander Song [2] Savir Basil [3]
Andy Huynh [4] Aneesh Raman [4]
Manos Athanassoulis [4]

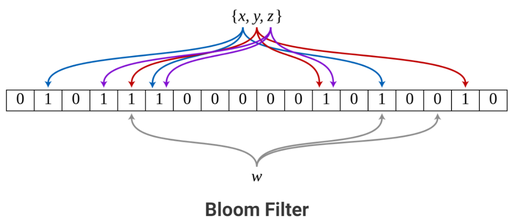
[1]Lycée Français de Vienne (Vienna, Austria) [2]High School (Sugar Land, TX)
[3]Sharon High School (Sharon, MA) [4]Boston University (Boston, MA)

Traditional Indexing Methods

Software such as MySQL or Oracle use *Traditional Indexing Methods*:



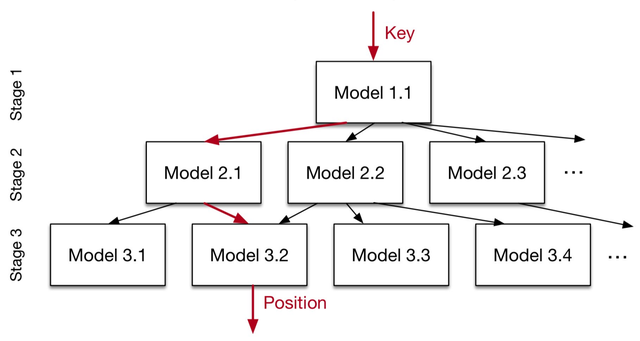
B-Tree



Bloom Filter

Learned Indexes

Recently, *Machine-Learning-based Indexes* were introduced to replace conventional indexes with the goal of improving performance and memory footprint [2]:



One such system is ALEX [1].

Introducing Sortedness

- K = number of out-of-order entries
- L = maximum displacement of any entry
- Example [3]:

References

[1] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. Alex: An updatable adaptive learned index. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 969–984, 2020.

[2] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 489–504, 2018.

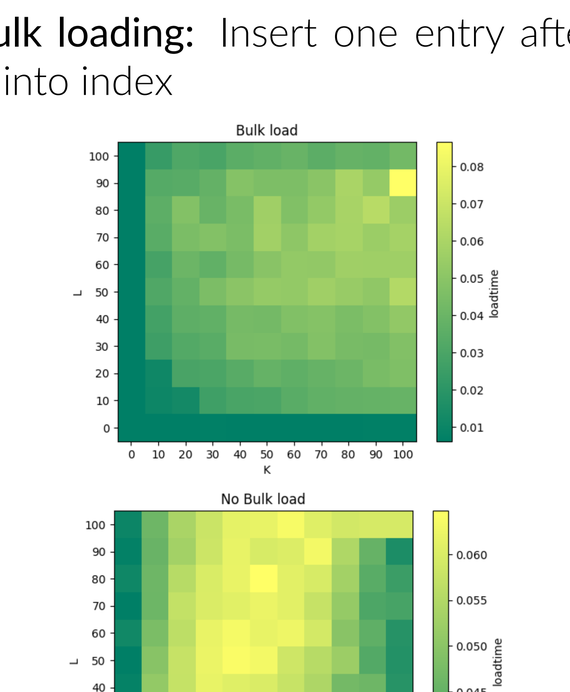
[3] Aneesh Raman, Konstantinos Karatsenidis, Subhadeep Sarkar, Matthaios Olma, and Manos Athanassoulis. Bods: A benchmark on data sortedness. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 17–32. Springer, 2022.

BenchLex

Our Goal: Evaluate how well ALEX indexes exploit sortedness.

- Built **BenchLex** = Python benchmark that measures the performance of ALEX for different values of L and K
- Created data consisting of 1 million entries with varying levels of sortedness using BoDS [3] with and without bulk loading
- Showed results in a heatmap of sortedness with respect to performance

Our Results



→ Efficiency decreases with sortedness

Loadtime

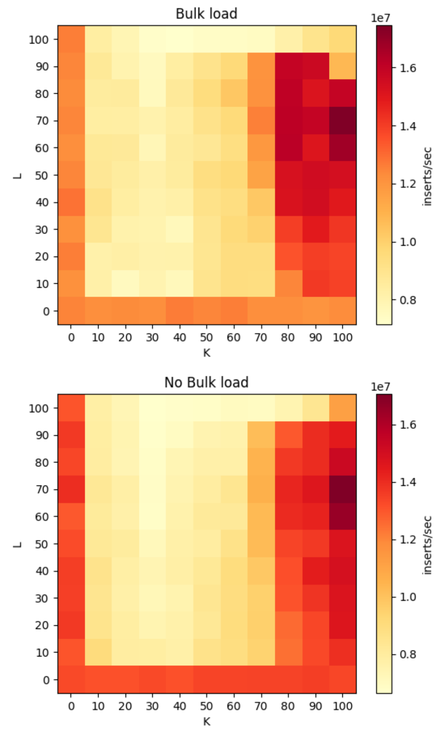
Time to load 1M entries into ALEX

Bulk loading: First sort all data and then build index from whole data at once

No bulk loading: Insert one entry after the other into index

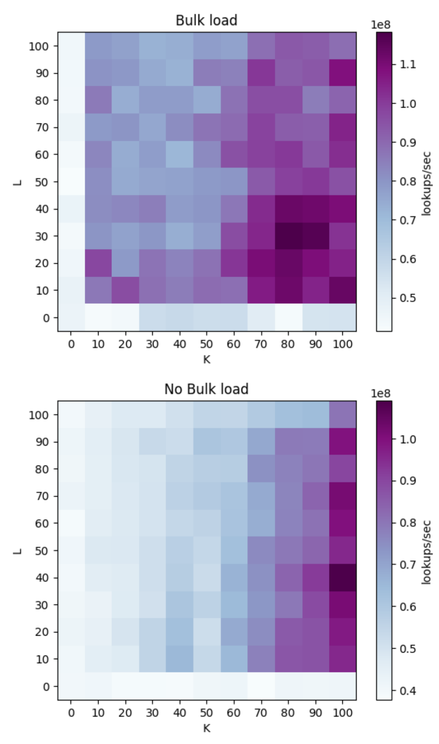
→ Worse performance without bulk loading

Inserts per sec



→ Poor performance for $K \leq 70$; very similar with and without bulk loading

Lookups per sec



→Poor when sorted, increases with K; generally better with bulk loading,

Conclusions

- Loading and insertions work well if the data is almost perfectly sorted or almost completely unsorted
- Lookups are poor when sorted and improve with degree of unsortedness
- Performance does not degrade in the same way with and without bulk loading
- Raises interesting questions about machine learning and randomization in general, and the performance of the ALEX system in particular