# Updatable Learned Index with Precise Positions
## Experiments

Wu J, Zhang Y, Chen S, et al., **2021 VLDB**

2024. 02. 14

Presentation by Nakyeong Kim, Suhwan Shin

nkkim@dankook.ac.kr, shshin@dankook.ac.kr

# Contents

# LIPP

**Entry type**
- ***NODE***: Pointer to a child node
- ***DATA***: Pointer to actual data
- ***NULL***: Unused entry space

## Learned Index with Precise Positions

- **Structure of Node**
  - Model (M): Predict position of the key
  - Entry array (E): Contain actual data points or pointers to child nodes
  - Bit vector: Identify entry type (e.g., ***NODE***, ***DATA***, ***NULL***)

- **Lookup Process**
  - Given a key, use top node's model(M) to predict position
  - The predicted position is checked against a bit vector for entry type
    - if entry type ***NODE***: Move to child node and repeat the process
    - if entry type ***DATA***: Access actual data via its data points

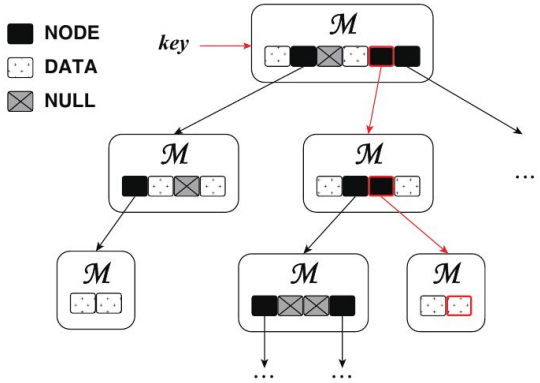


Figure 2: Structure of LIPP

- **Summary**
  - Accurate position prediction means that will insert it anyway, despite the conflict
  - FMCD helps to select a model minimizing conflicts

# Definition

## Term

- **Error**: Failure to predict positions for keys in the model
- **Conflict** (Collision): Predict the same position for other keys
- **Height** (Depth): Depth of the deepest node
- **Range Query**: Operation of getting N values from lower key
- **Space Amplification**: Ratio between space used to store data and the size of the original data
  - *SAF = used space / total space*
  - Closer to 1 is more efficient

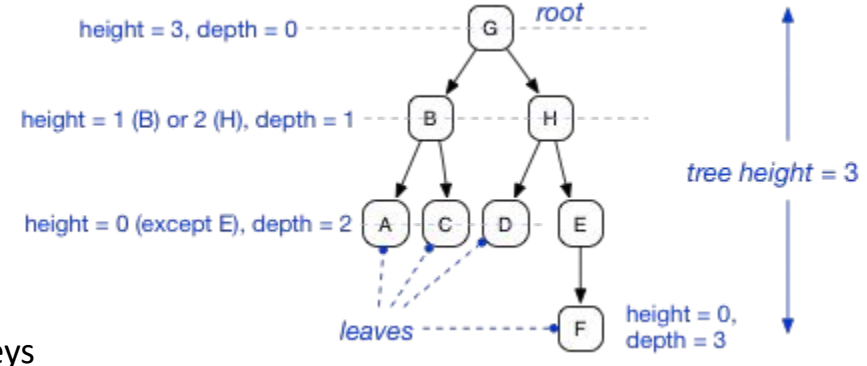
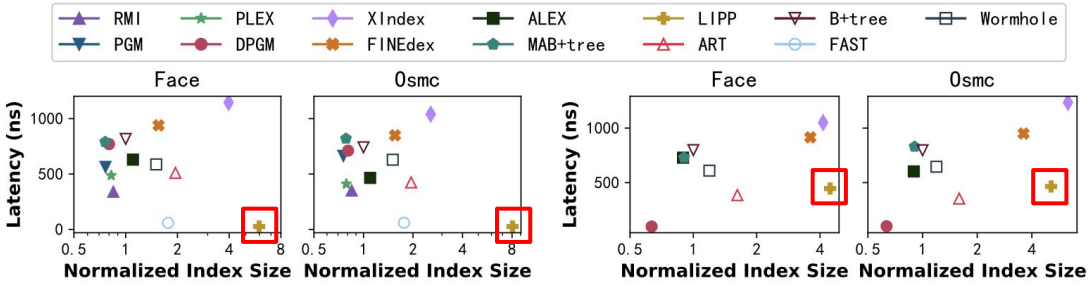



**Figure: Structure of Tree**

# Motivation

## Motivation of Experiment




- **Problems with LIPP**
  - Not tolerate errors
  - Create child nodes when conflict occurs (conflict-based structural modification)
  - The more conflicts ⟶ the higher height of tree ⟶ **space amplification**
  - Violates the space efficient principle of learned index

- **Goal**
  - Analyze the impact of space amplification due to conflicts
  - Try to solve it
  + Also, analyze the performance of range query

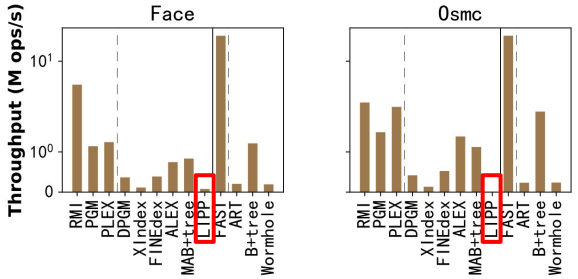



**Figure 8: Throughput of range queries. The solid line divides the chart into learned indexes (left) and traditional indexes (right), for which the dotted lines separately divide into immutable (left) and mutable indexes (right).**

# Experiment

## Settings

- **Datasets**
  - simple
    - books (Amazon)
    - history
  - dynamic
    - fb (Facebook)
    - osm

- **Common parameters**
  - table_size: 1,000,000
  - scan_num: 100
  - read : insert : scan = 0 : 0 : 1
  - thread_num: 1
  - do operations after bulk loading (table_size)

```
processor       : 19
vendor_id       : GenuineIntel
cpu family      : 6
model           : 151
model name      : 12th Gen Intel(R) Core(TM) i7-12700K
```

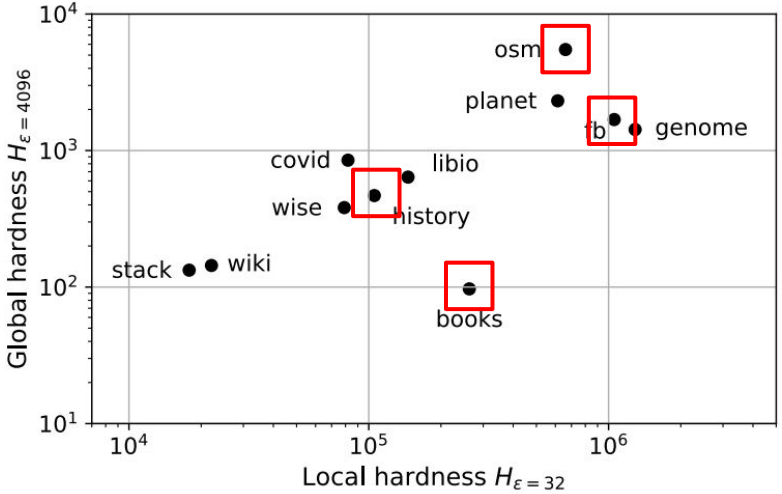


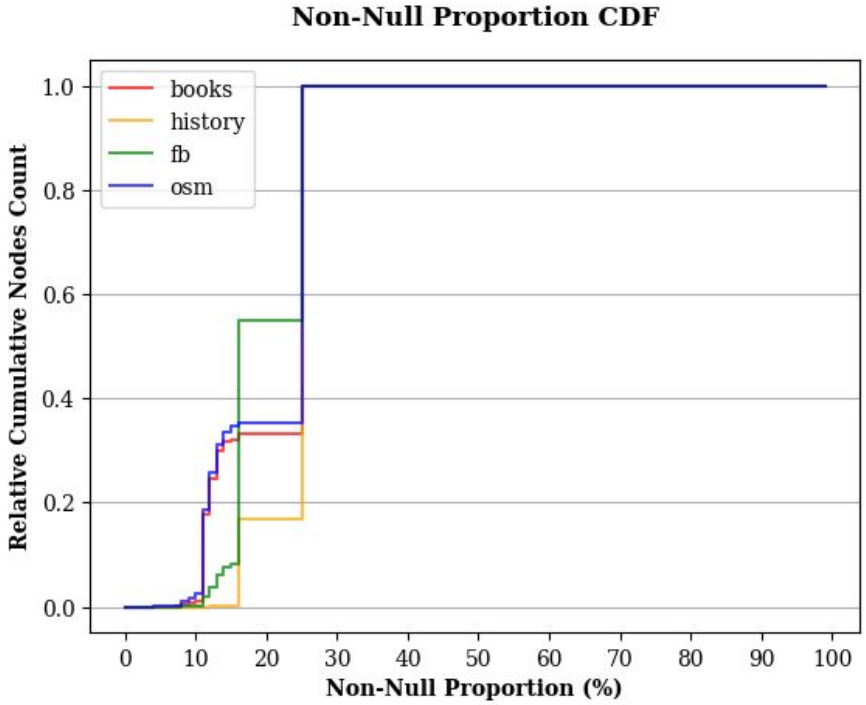

**Figure 5: Datasets used in the experiments.**

# Experiment 1

Normalize ***Data***, ***Node*** Entries per node to the total node

## Distribution of Node's Utilization

- To observe the ratio of **entries actually being used** for each node
- **History** uses space most efficiently, while **fb** not
- Most nodes have less than 30% of all entries (high space amplification)



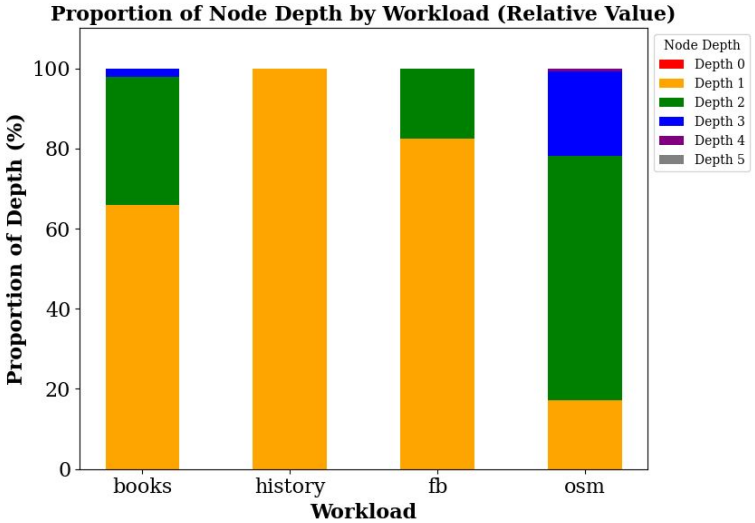
# Experiment 2

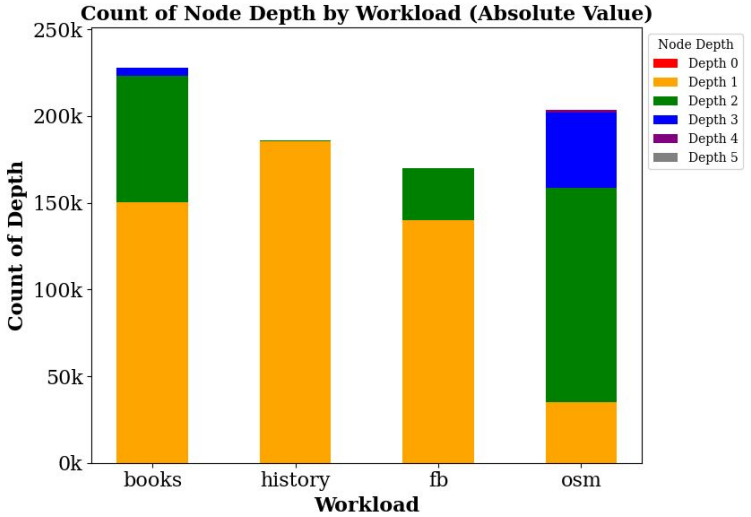
## Distribution of Node's Depth

- To observe relation between node depth distribution and workloads
- Higher maximum node depth indicates that more conflicts occurred

| Depth | books | history | fb | Osm |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 |
| 1 | 150189 | 185744 | 140051 | 35123 |
| 2 | 73300 | 181 | 29968 | 123687 |
| 3 | 4654 | 0 | 4 | 43046 |
| 4 | 1 | 0 | 0 | 1583 |
| 5 | 0 | 0 | 0 | 4 |

**Figure: Count of Node Depth (Absolute)**



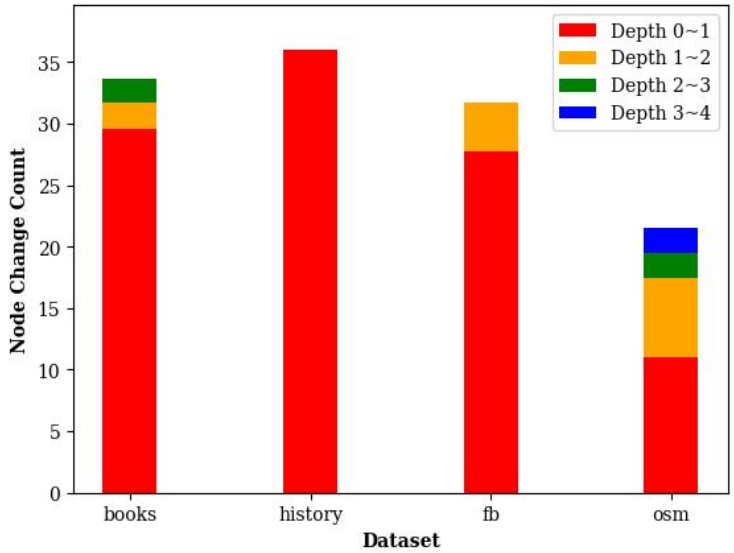



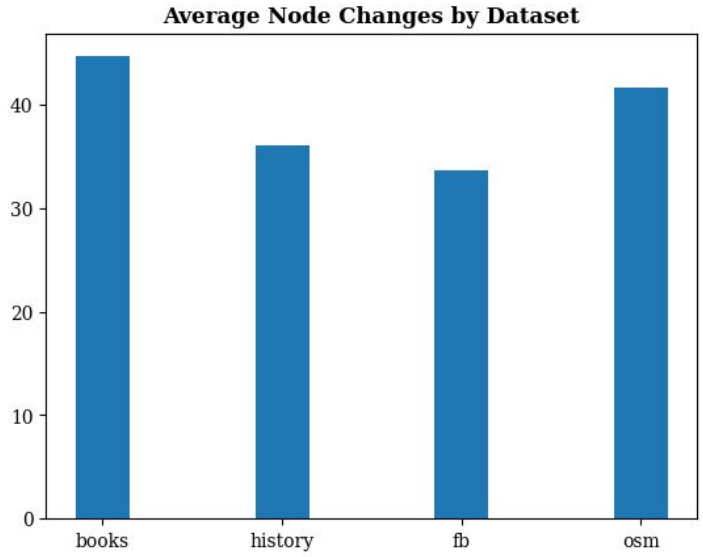
# Experiment 3-1

**NNP**: Non-Null Proportion

## A Number of Node Change (Depth)

- To observe the overhead of range query
- Based on the results of experiment 1, expected the smaller **NNP**, the more nodes are searched

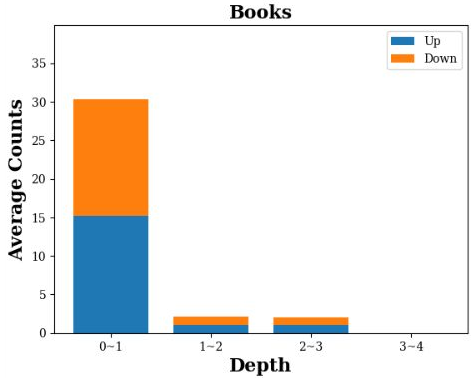


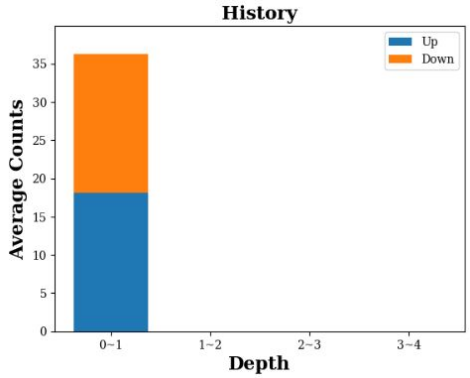



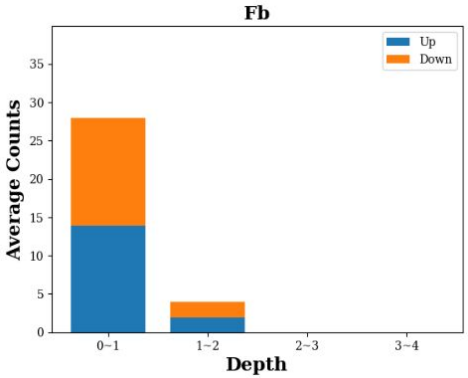
# Experiment 3-2

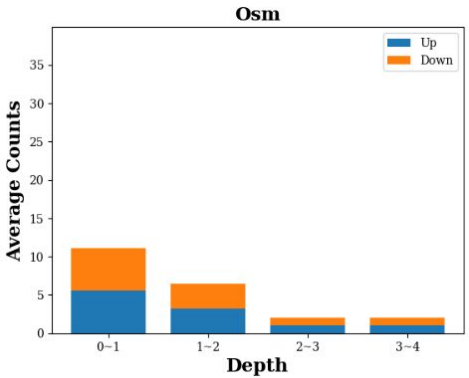
## A Number of Node Change (with Up/Down)

- Have almost similar up/down count



# Next Step

## Error-Tolerance Strategy

If allow error, there will be fewer conflicts and thus less space amplification.

- **How much** tolerance for errors?
- **How to** tolerate errors?
  - If a conflict occurs, create it as a sibling node instead of a child node (array or list)
  - …

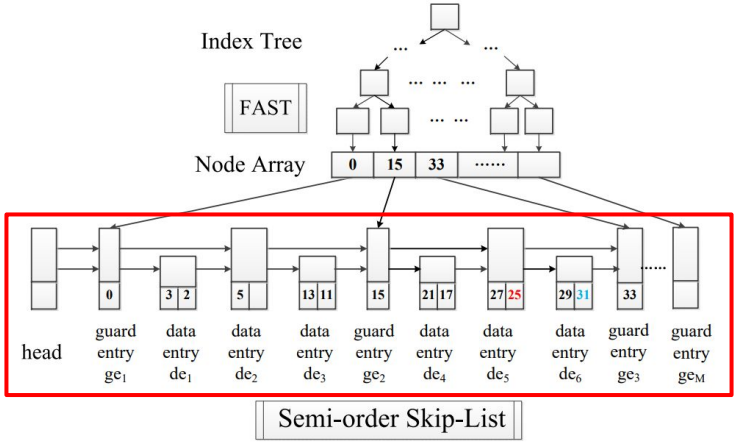



**Figure: Architecture of S3**

# Conclusion

## Conclusion

- LIPP had a **large index size** compared to the other indexes. We suspected that this was due to **space amplification** caused by the increased height of the tree due to conflicts, and wanted to verify this

- We measured node utilization, height distribution and node count in range query. We found that SAF wastes more memory than expected, and that many node changes occur during range queries

- We want to solve the problem with LIPP by allowing some errors, or in some other way

# Q&A

# Thank you!

# Experiment 3-2

## A Number of Node Change

- To observe how many node changes on average to perform a range query once