Updatable Learned Index with Precise Positions
Experiment

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

2024. 02. 28

Presentation by Nakyeong Kim, Suhwan Shin
nkkim@dankook.ac.kr, shshin@dankook.ac.kr

Contents

1. Introduction

1) Motivation

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
- <u>Try to solve it</u>

\+ Also, analyze the performance of range query

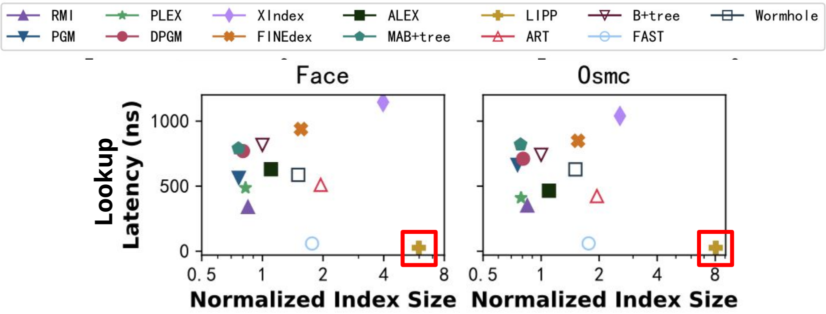


Figure 1: Trade-off of performance and normalized index size.

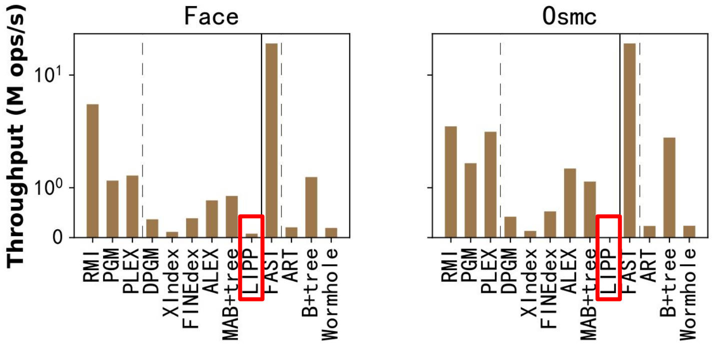


Figure 2: Throughput of range queries.

1. Introduction

Node Utilization: Proportion of non-null entries.

2) Observe with a focus on Utilization

Most nodes have less than 30% of all entries

→ There exists upper bound

$$T_{\mathcal{M}} = \max_{l \in [0, L-1]} |\{k \in \mathcal{K} | \mathcal{M}(k) == l\}|$$

We observe that there exists an upper bound for the minimum $T_{\mathcal{M}}$, i.e. $\exists \mathcal{M}, T_{\mathcal{M}} \leq \lceil \frac{N}{3} \rceil$ where N is the number of keys in $\mathcal{K}$, i.e. $N = |\mathcal{K}|$. However, the $\lceil \frac{N}{3} \rceil$ may not be the tightest upper bound in many cases. Thus, our goal is to find a best model $\mathcal{M} = A\mathcal{G}(k) + b$ with the minimum conflict degree $T_{\mathcal{M}}$.

We think that **TM** will be an important factor of space amplification

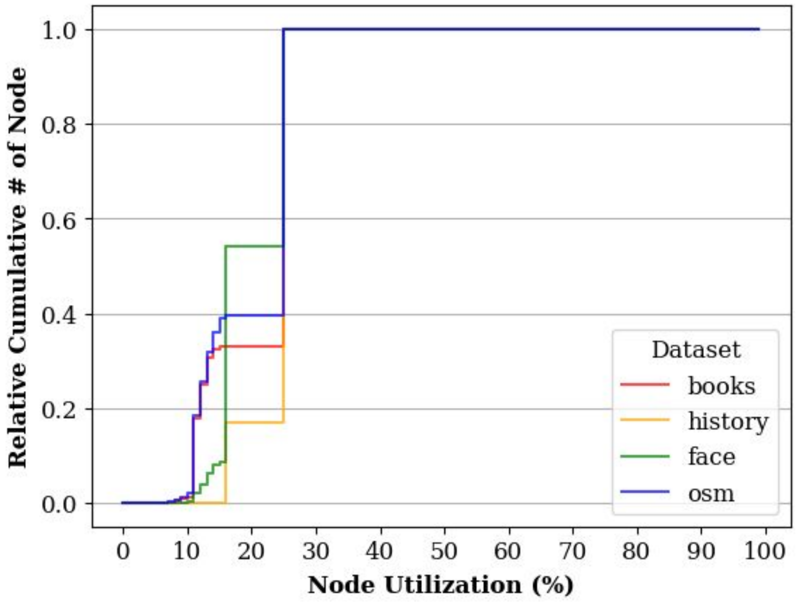


Figure 3: Node utilization CDF

1. Introduction

$$T_{\mathcal{M}} = \max_{l \in [0, L-1]} |\{k \in \mathcal{K} | \mathcal{M}(k) == l\}|$$

3) Parameters of LIPP

- We said that our observation, node utilization upper-bound, is caused by TM, but it is not true

- We found that the factor of affecting to utilization is fill factor(initial node size, gap count)

- We assume that controlling this factor will change performance and index size

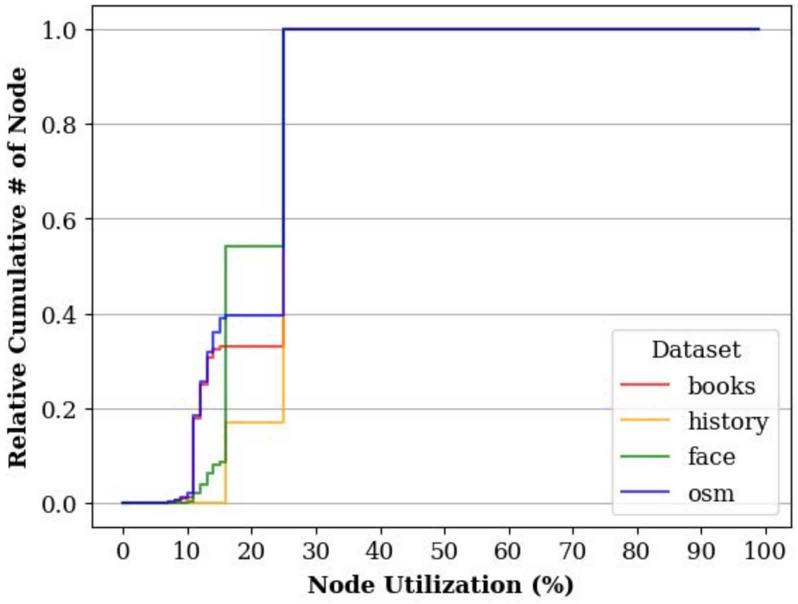


Figure 3: Node utilization CDF

1. Introduction

Fill factor: How many fill space when given new keys, inverse of utilization
e.g., if fill factor is 2, node utilization is 1/2

4) To do Works

1. A sensitivity analysis into updatable learned index structure
   - Node utilization management policy: ***fill factor***
   - Model: simple linear regression vs kernelized linear regression
   - Conflict resolving: shifting vs chaining
   - SMO(Structural modification operation): cost-benefit(fanout tree) vs conflict-proportion

1. Introduction

4) To do Works

2. Performance comparison between ALEX and LIPP through size
   - Need understanding of fill factor(parameter) each indexes

3. Which techniques are appropriate when considering performance versus space?
   - Create a new index based on that analysis
     - + Conflict resolving : error-controlled approach (shift-chain hybrid)
     - + Concurrency-friendly : semi-ordered

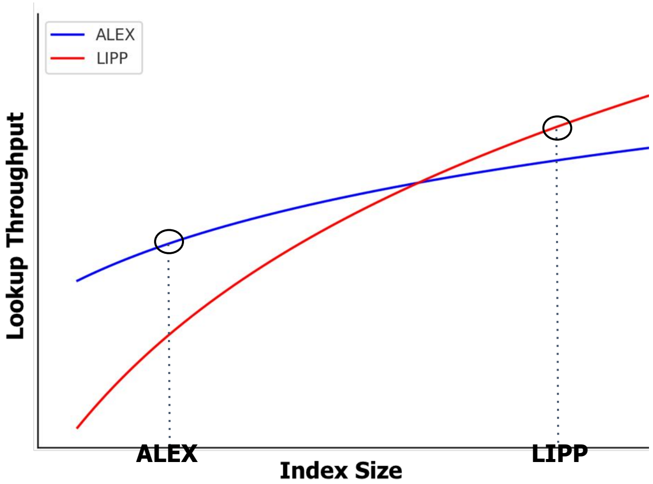


Figure 4: Expected lookup throughput according to size.

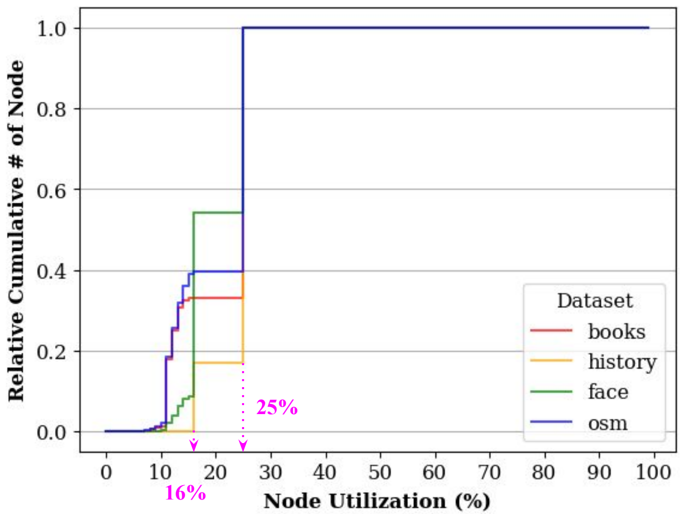
2. Observations

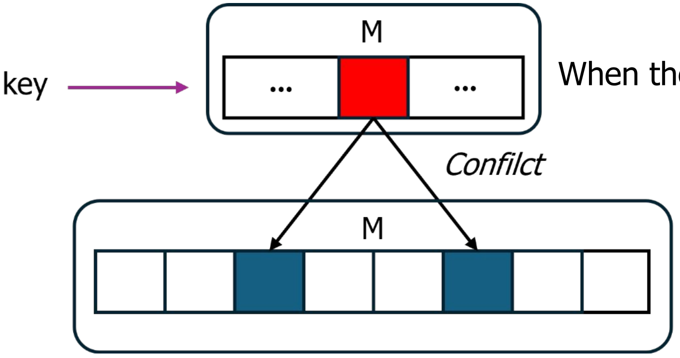
1-1) Node Utilization

Node size set when

(1) Build tree at first (bulk load)

(2) Rebuild (adjust)





When the entry is already ***Data*** type

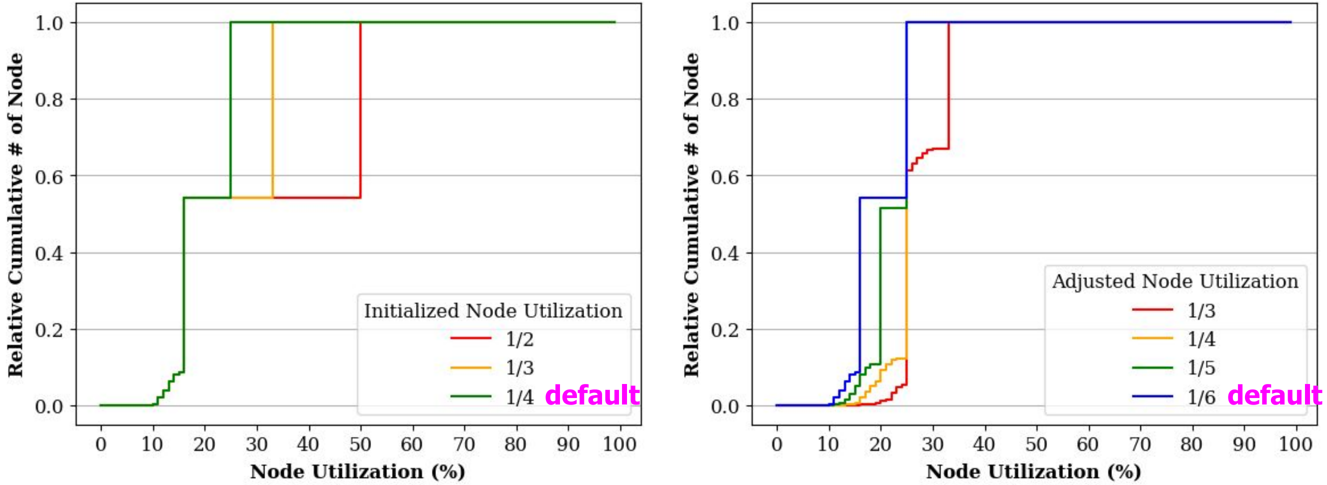
→ Utilization: 2/8 (initial node size) = 25%

Figure 5: Making "Node" type entry when conflict

2. Observations

Observed after bulk load **100M** keys
Dataset: **Face (1.6GB)**

1-2) Node Utilization CDF



- The initial node size greater, the utilization lower
- The gap size greater, the utilization lower

2. Observations

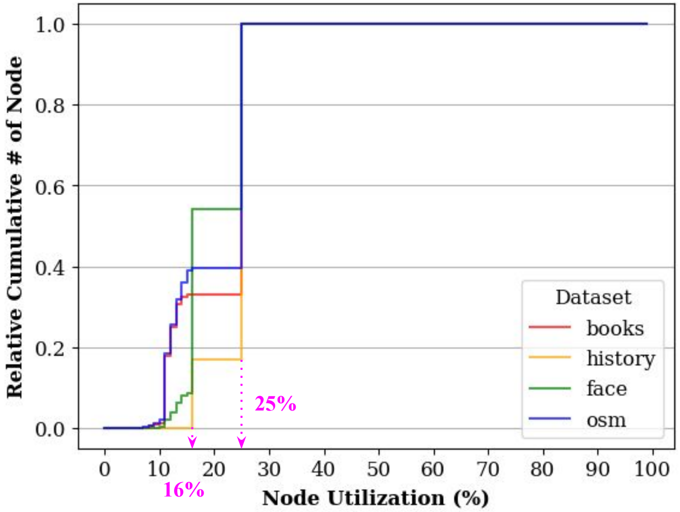
2) Static Gap Count

- It is important to determine **the array length of new node**, because of trade-off between performance and space consumption

- When new node is first created, gaps are hard-coded with **1,2,5** depending on the keys size
    - 1, size ≥ 1_000_000
    - 2, size ≥ 100_000
    - 5, default

- Almost all nodes have a count lower than 100_000, which means that on average, <u>⅙ utilized</u>

- It has low hotness

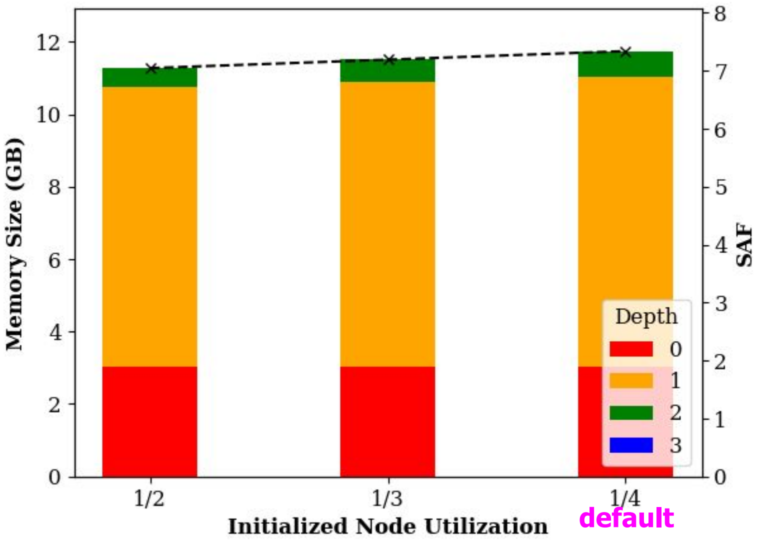


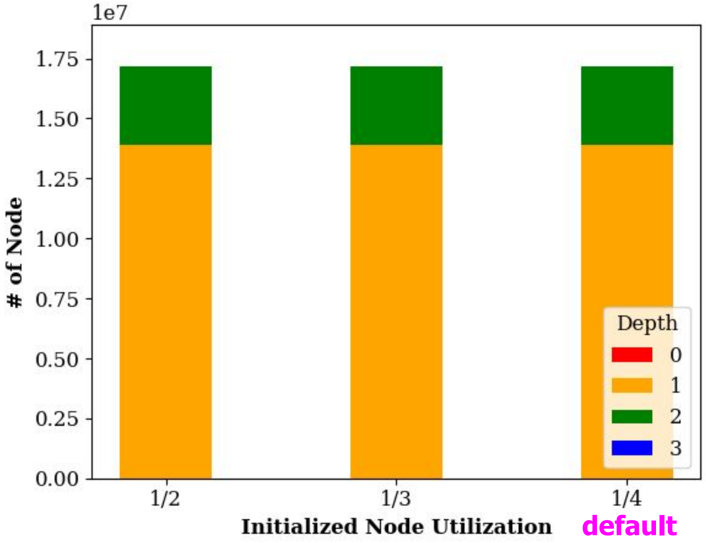

```
const int BUILD_GAP_CNT = compute_gap_count(size);

node->is_two = 0;
node->build_size = size;
node->size = size;
node->fixed = 0;
node->num_inserts = node->num_insert_to_data = 0;

{
    const int L = size * static_cast<int>(BUILD_GAP_CNT + 1);
node->num_items = L;
```

2. Observations

Observed after bulk load **100M** keys
Dataset: **Face (1.6GB)**

3) Root Node Size (Initial Node Size)



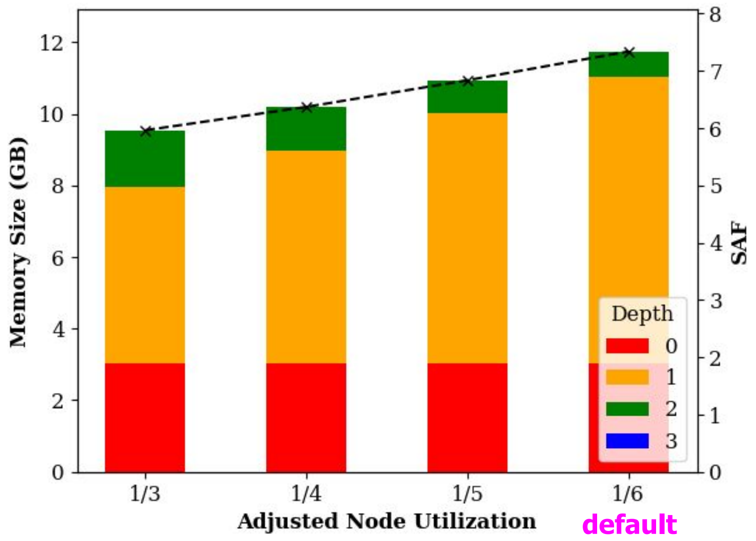


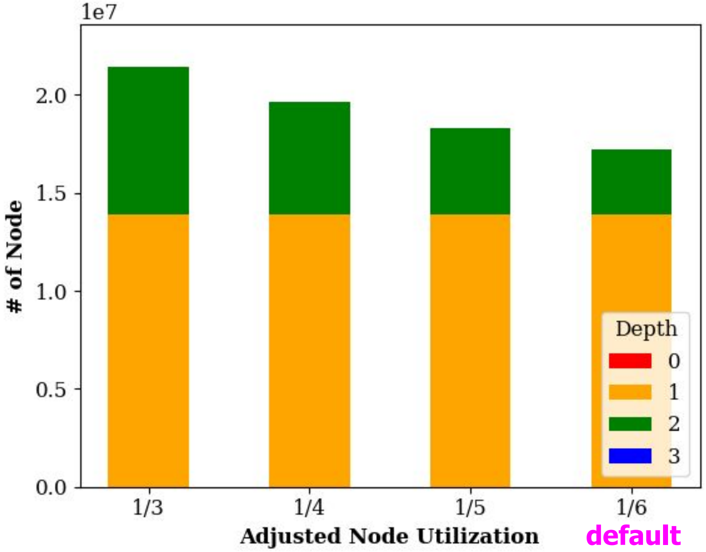
- Root node takes up a significant percentage of the total index size
- Initial node size doesn't have much impact on overall size
- Large number of nodes, but small percentage of size (MBs)

2. Observations

Observed after bulk load **100M** keys
Dataset: **Face**

3) Root Node Size (Gap Count)

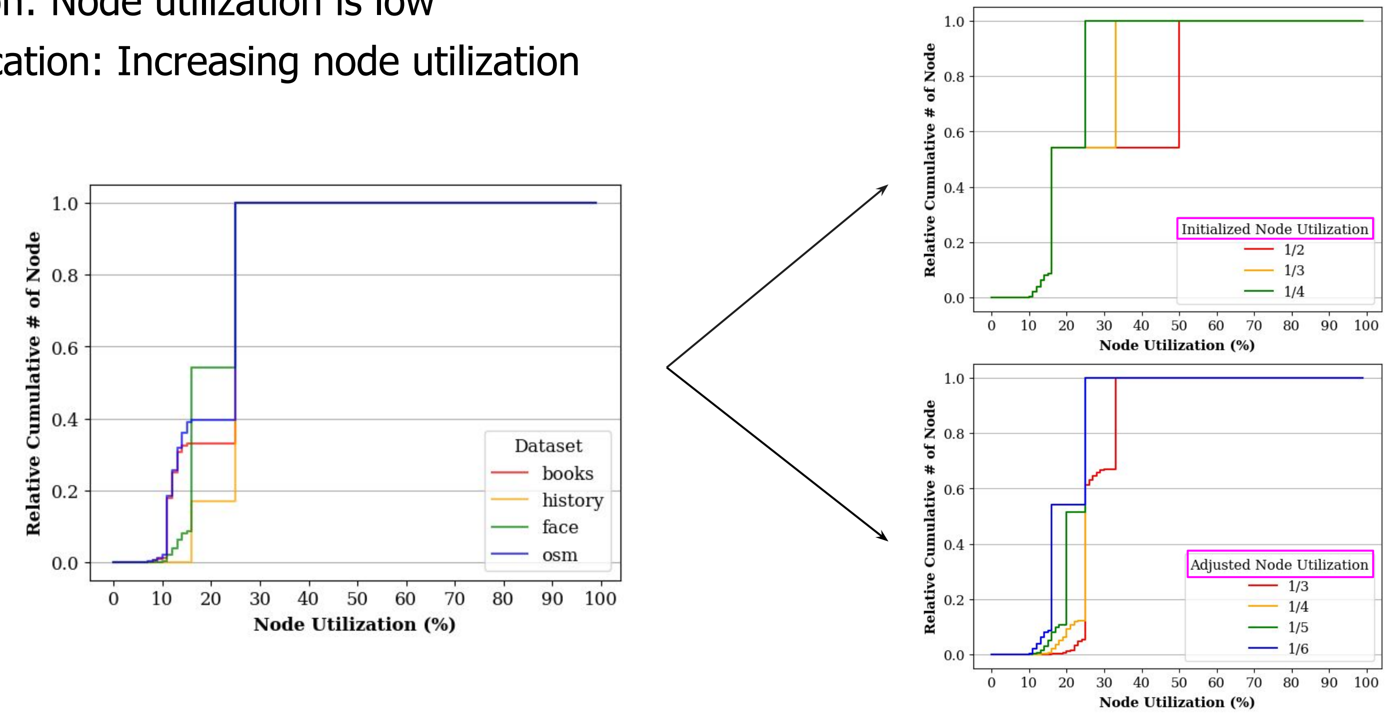




- Gap count has more impact than initial node size, but not primarily
- There may be some other factors (e.g., α, β, FMCD …)

3. Hypothesis

- Hypothesis: The cause of LIPP's SAF may be <u>array management policy</u> (node utilization)
- Reason: Node utilization is low
- Verification: Increasing node utilization

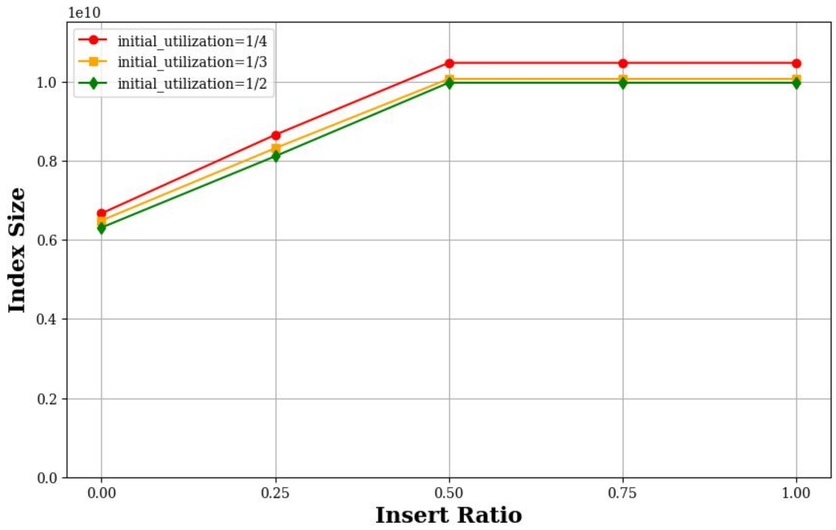


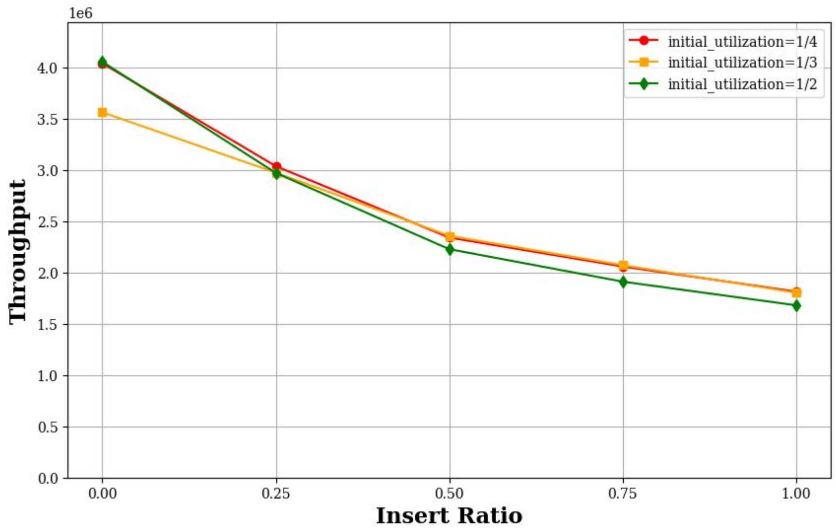
4. Experiments

Observed after bulk load **100M** keys
Dataset: **Face**

Goal : Impact of lipp's array utilization policy (initial node size)

Observation. The **initial node size** has little effect on index size and read-write performance



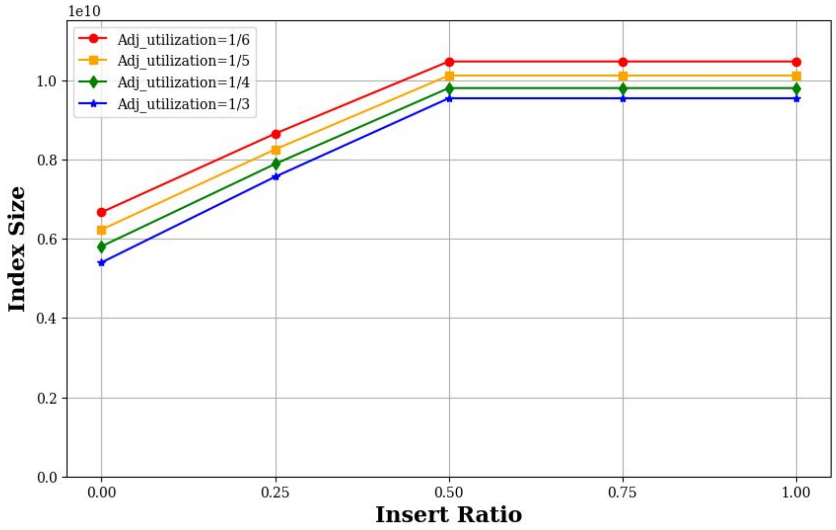


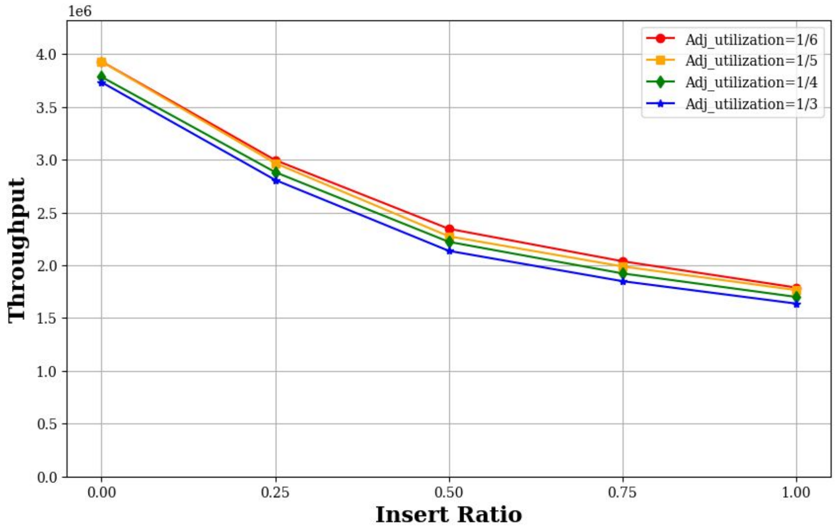
4. Experiments

Observed after bulk load **100M** keys
Dataset: **Face**

Goal : Impact of Lipp's array utilization policy (gap count)

Observation. The **gap count** has a greater impact than the initial node size, but it is not the main cause





5. Conclusion & Question

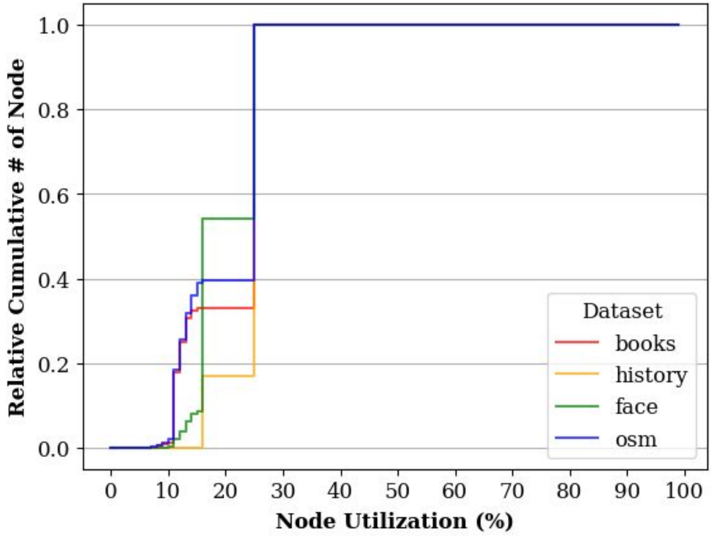
Experiment conclusion:

- Array management policy was not the main cause of Space Amplification (SAF)

Question: No nodes exceed initial node utilization (25%)

- If fill factor is the problem, 3/8 should also exist!
- But, node with a utilization rate of 3/8 is not observed

→ Probably FMCD, conflict issue or something…



6. Future work

1. A sensitivity analysis into updatable learned index structure
- **Current Hypothesis: Array management policy**
    - **fill factor: initial node size, gap_count**
- **New Hypothesis (Expectation)**
    - **rebuilding condition: α, β**
    - **training model: FMCD**
- **Model : simple linear regression vs kernelized linear regression**
- Conflict resolving : shifting vs chaining
- SMO (Structural modification operation) : cost-benefit (fanout tree) vs rebuilding

$$\frac{n.element\_num}{n.build\_num} \geq \beta \quad \beta \text{ is set to 2 by default}$$

$$\frac{n.conflict\_num}{n.element\_num - n.build\_num} \geq \alpha \quad \text{we set the threshold } \alpha = 0.1$$

2. Performance comparison between ALEX and LIPP through size
- Need understanding of fill factor parameter each indexes

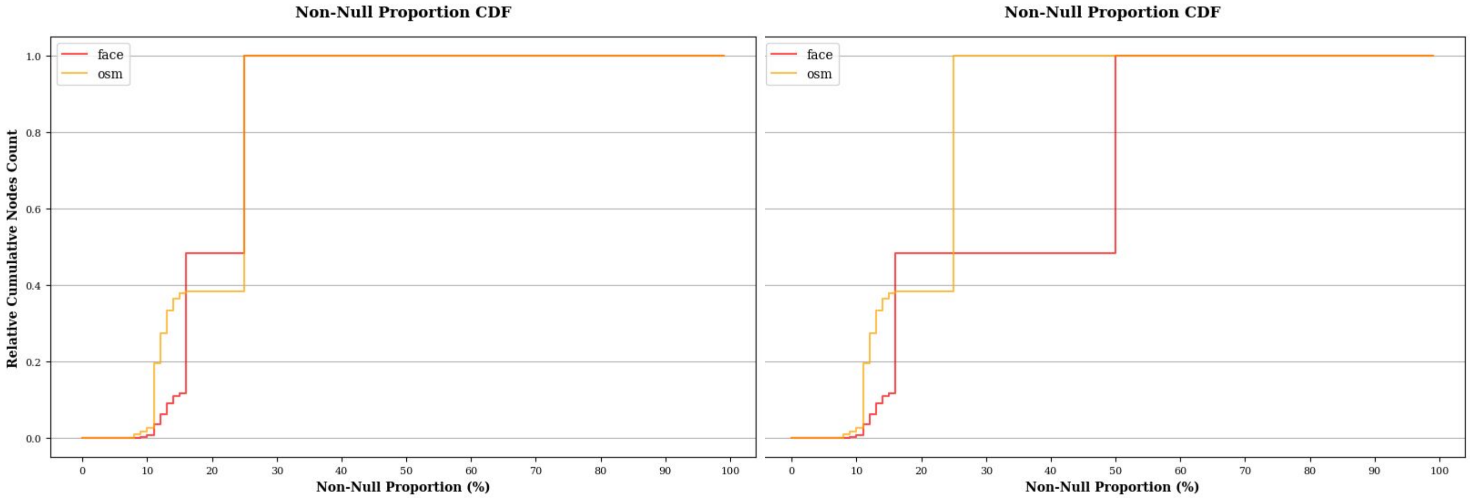
3. Which techniques are appropriate when considering performance versus space?
- Create a new index based on that analysis
+ Conflict resolving : error-controlled approach (shift-chain hybrid)
+ Concurrency-friendly : semi-ordered

Q&A

Thank you!

2. Observation

3) Ro



4. New? Hypothesis

- Adjustment trigger

  1) fill factor

  2) conflict num

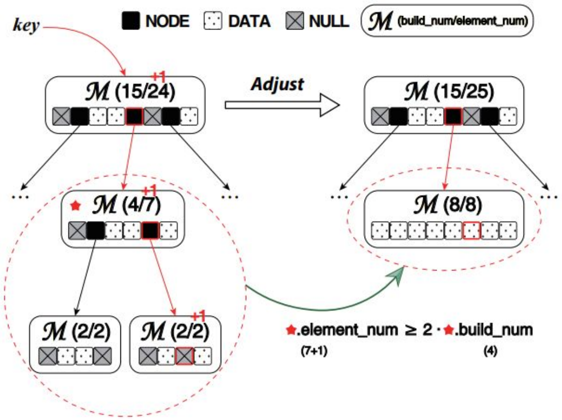


Figure 4: Node Adjustment

- This is because we are considering the utilization of the sub-tree, not the node
- When β is 2, the actual utilization may be much less than 50%

$$\frac{n.element\_num}{n.build\_num} \geq \beta \quad \beta \text{ is set to 2 by default}$$

$$\frac{n.conflict\_num}{n.element\_num - n.build\_num} \geq \alpha \quad \text{we set the threshold } \alpha = 0.1$$

1. Summary

Motivation

Poor understanding of Updatable Learned Index (LIPP)

→ Additional studying: Array Size Policy, Rebuilding Process, FMCD

Analysis of relationship between **index size** and **performance**

→ Between LIPP and ALEX, which structure is more suitable for a learned index?

+ Error-Controlled Approach, Range Query, Semi-Ordered

1.1. Adjustment Strategy

When to Adjust

1. Insert key(s)

2. Update and check statistics of nodes in the traversal path

3. Trigger adjustment on a chosen node when certain conditions are satisfied
   a. $\frac{n.element\_num}{n.build\_num} \geq \beta$ β is set to 2 by default
   b. $\frac{n.conflict\_num}{n.element\_num - n.build\_num} \geq \alpha$ we set the threshold $\alpha = 0.1$

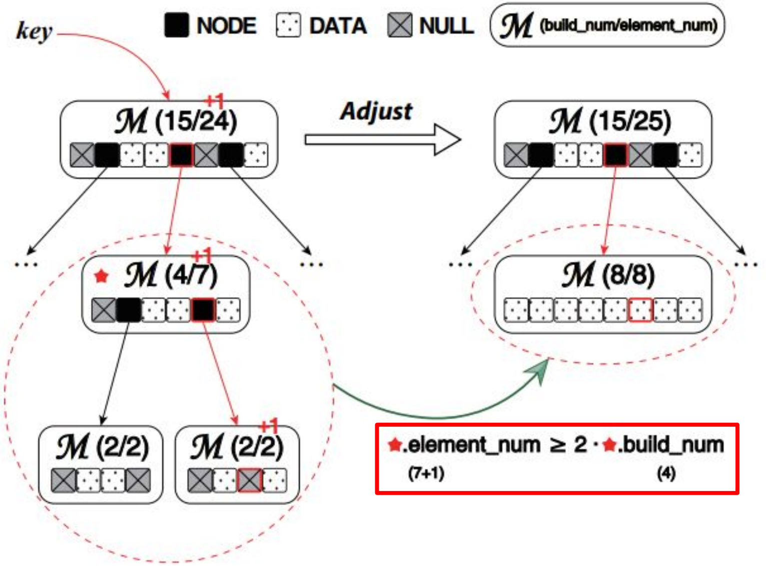


Figure 4: Node Adjustment

1.1. Adjustment Strategy

How to Adjust

1. Collect all elements(keys) in the subtree rooted at node by sequential traversal

2. Build a partial tree on the elements

3. Update the pointer of the original node to point of the new node(tree)

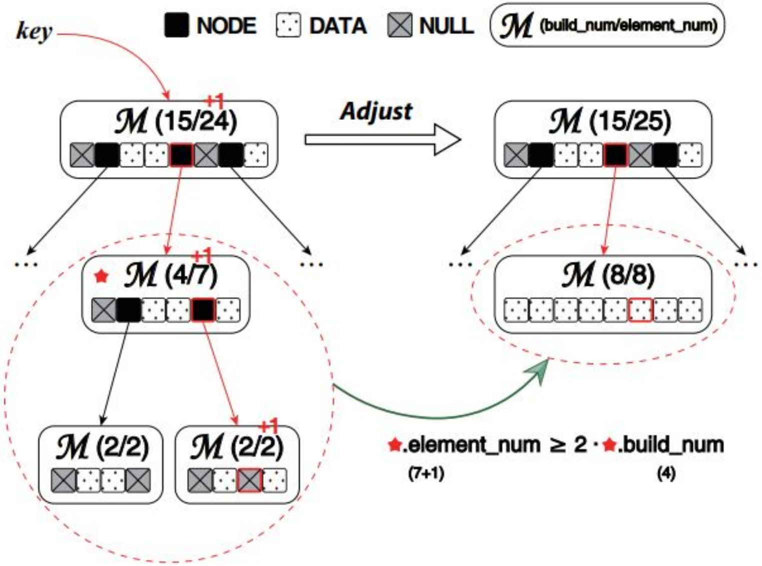


Figure 4: Node Adjustment

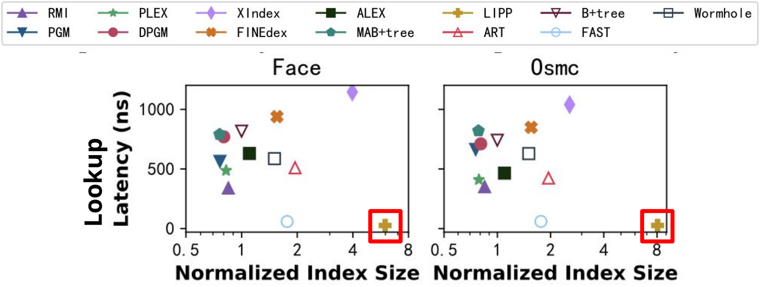
1. Summary

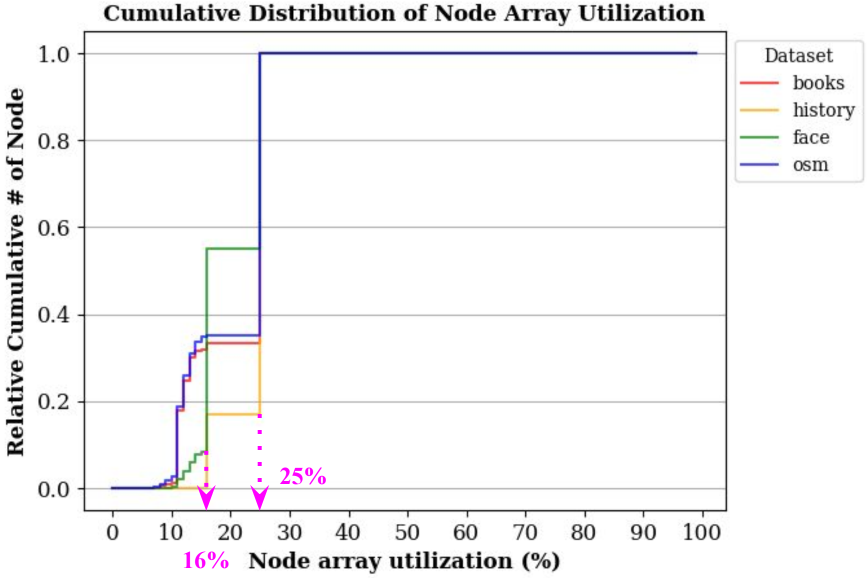
Parameter of Index

Most of raw data is “8,2”, which means “total,node” entries.

There are things the paper doesn't show.

- Array size of child node when first created is hard-coded by 8. (0.25 utilization)
- Array size of new node when need rebuilding(or bulk loading) can be 2x, 3x, and 6x, respectively.





1. Summary

Purpose

Lookup/Insert performance comparison according to size

- Tuning parameter: ALEX(fill factor), LIPP(gap count)

LIPP vs ALEX

- Model
- Array Management Policy
- Shift/Chaining
- SMO

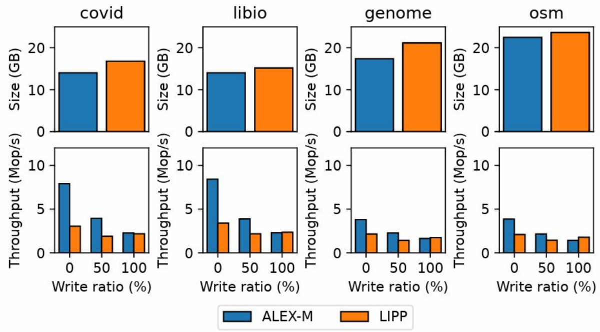
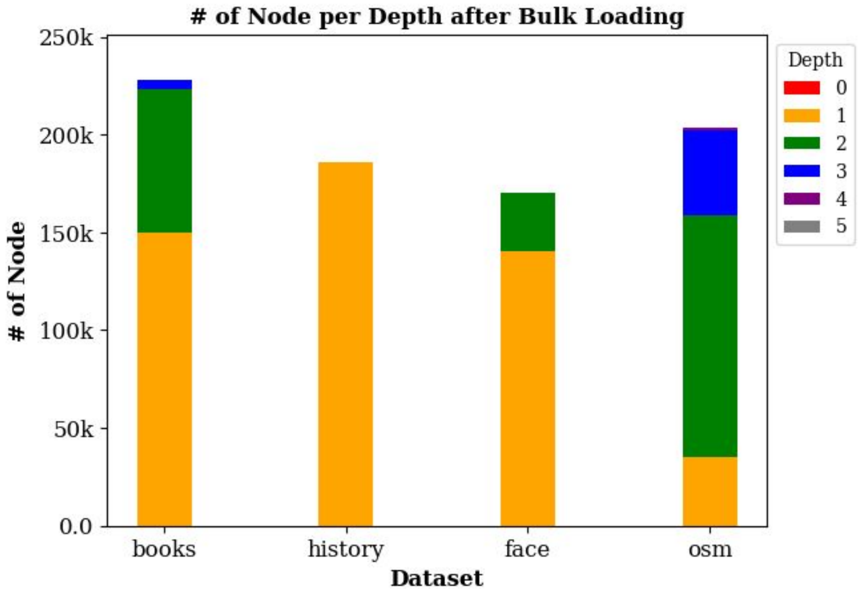
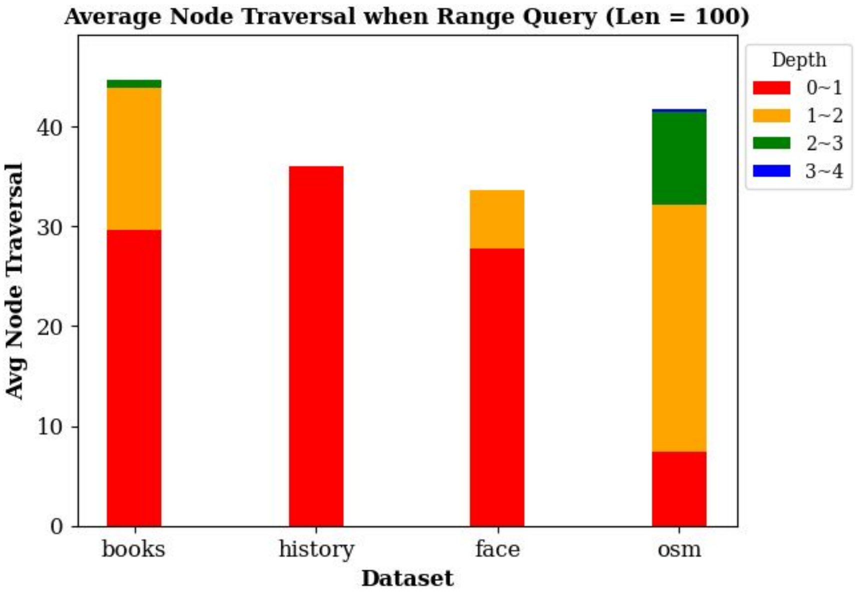


Figure 9: ALEX-M vs. LIPP (when ALEX is tuned to use roughly the same amount of memory as LIPP).

2.1. Experiment

Tendency by Dataset

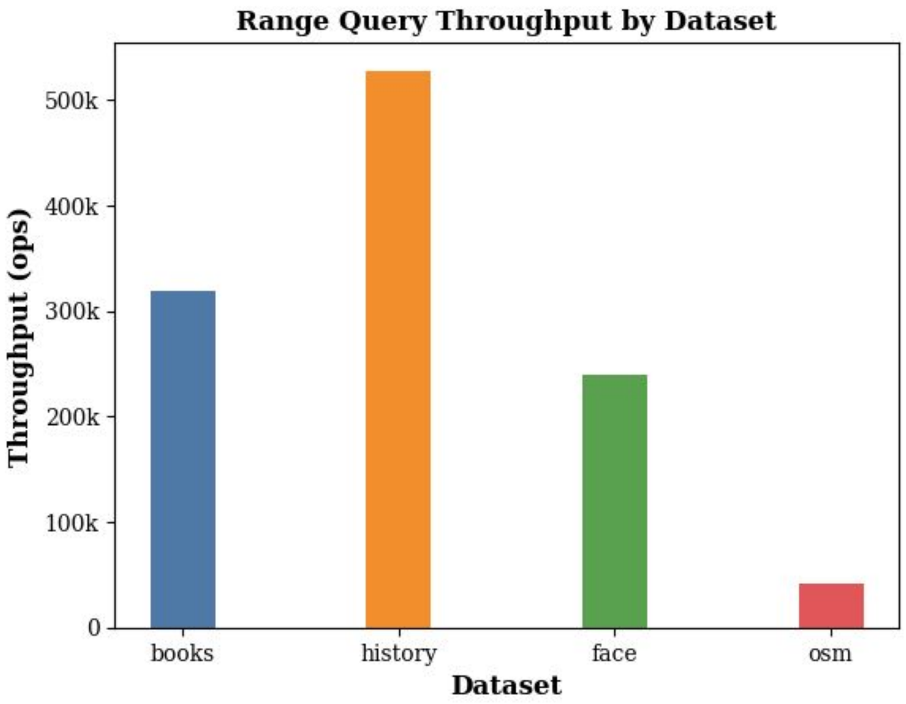




2.2.1. Experiment

scan_num: 100
iteration: 1M
table_size: 100M (key_range: 200M)
operations: range query only

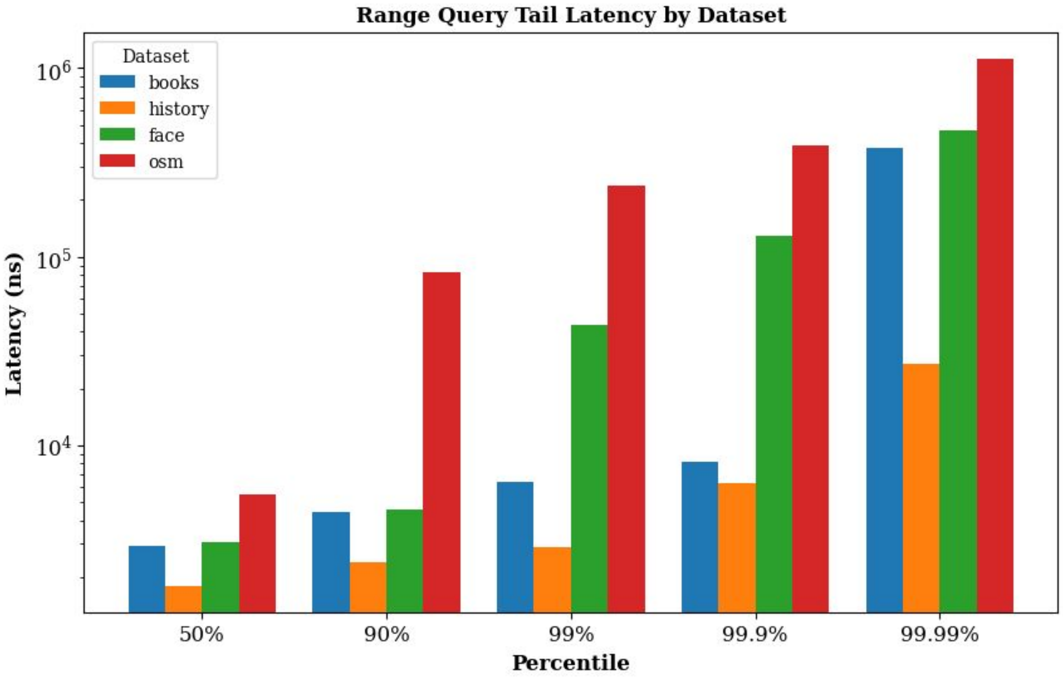
Range Query Throughput by Dataset



2.2.2. Experiment

scan_num: 100
iteration: 1M
table_size: 100M (key_range: 200M)
operations: range query only

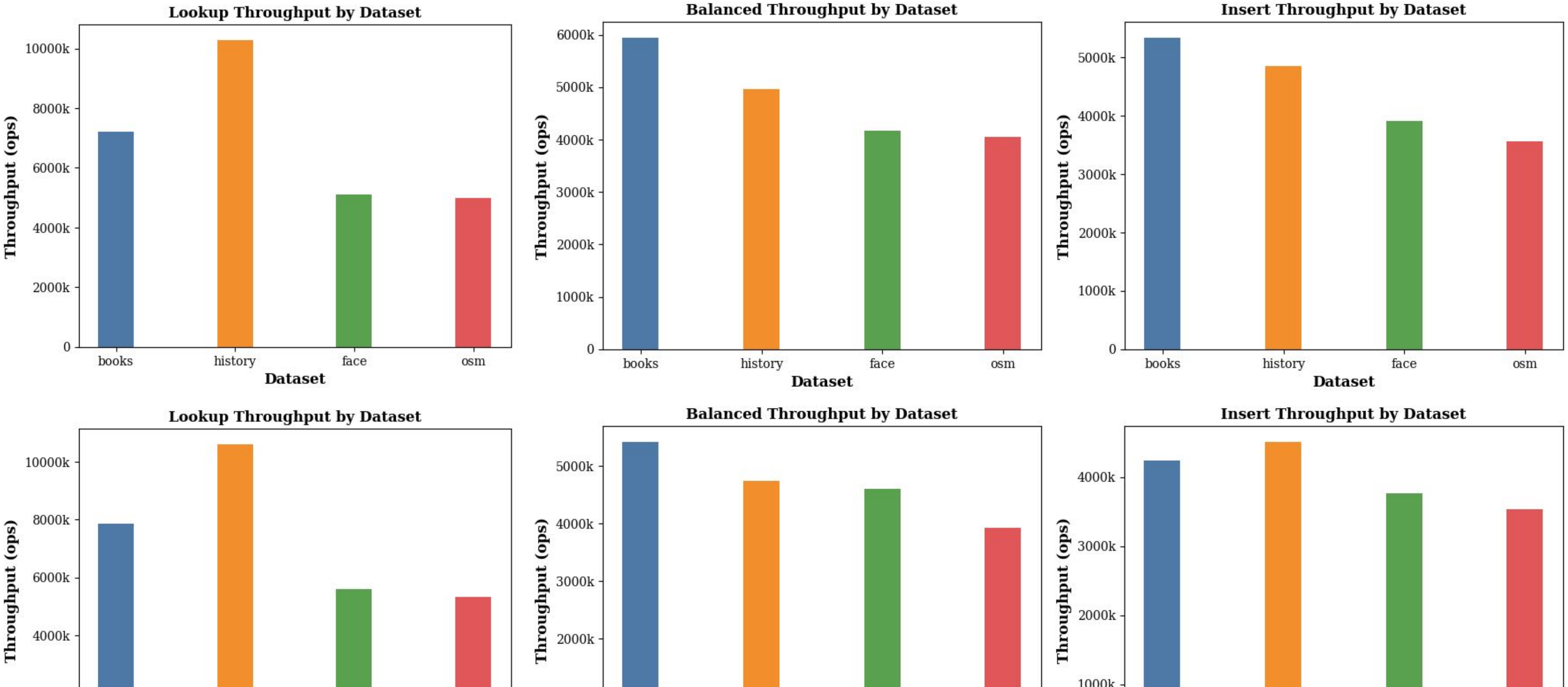
Range Query Latency by Dataset



2.2.2. Experiment

scan_num: 100
iteration: 1M
table_size: 100M (key_range: 200M)
operations: range query only

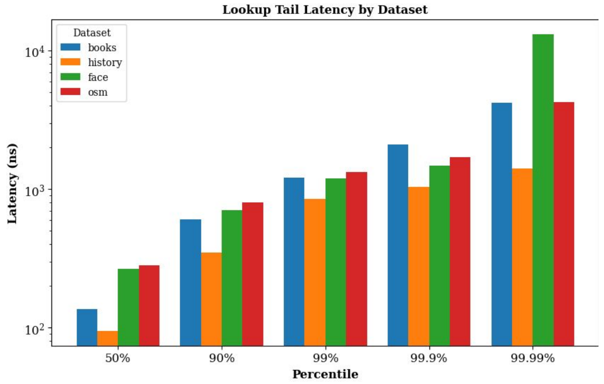
Lookup Throughput by Dataset

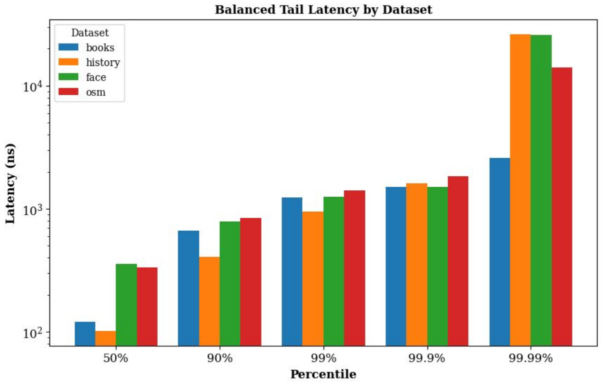


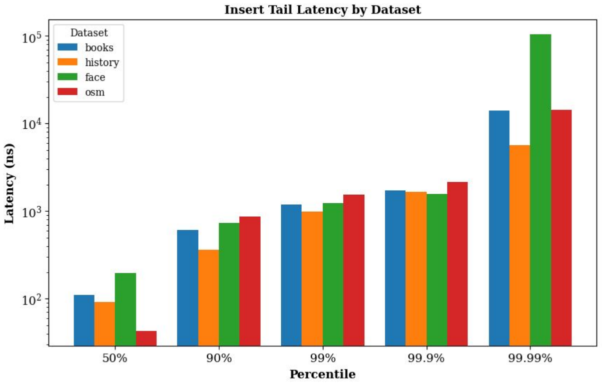
2.2.2. Experiment

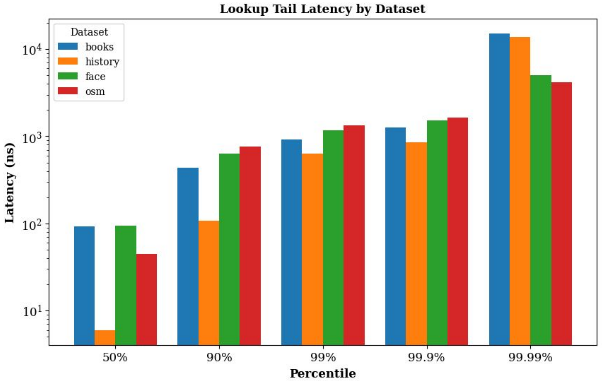
scan_num: 100
iteration: 1M
table_size: 100M (key_range: 200M)
operations: range query only

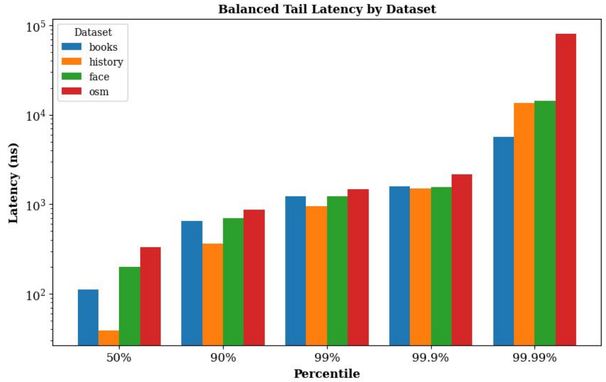
Lookup Latency by Dataset

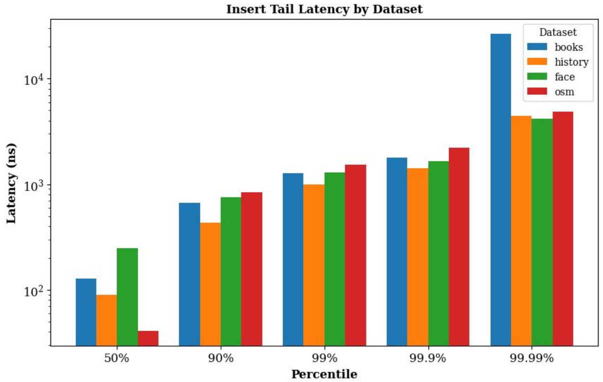












2.3. Dataset Hardness

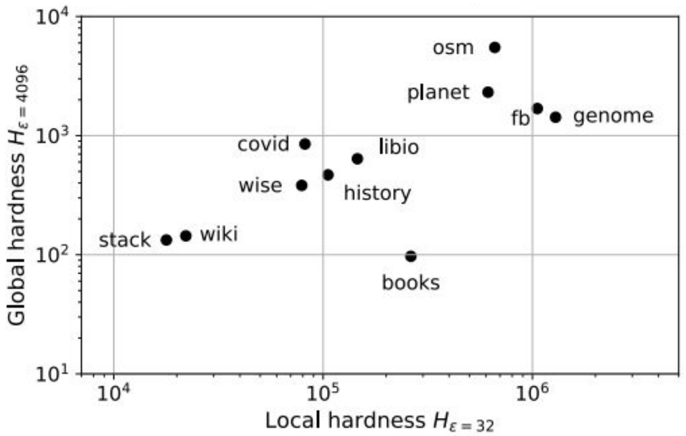
New Criteria

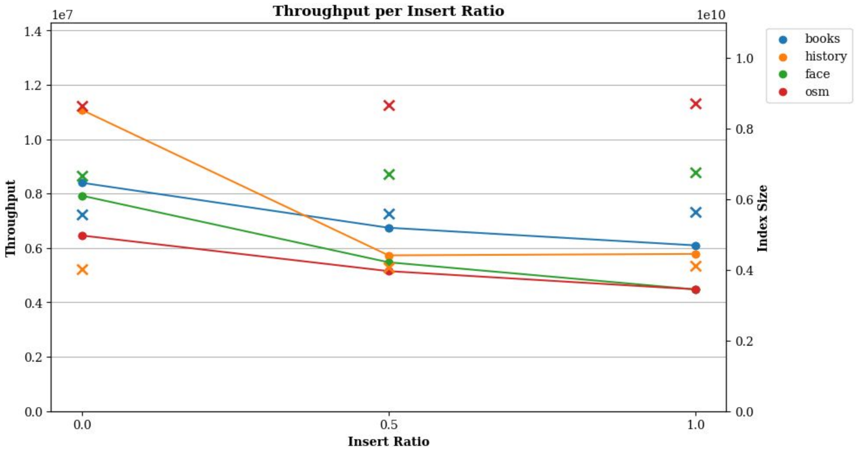
- Previous experiments defined data hardness as the number of segments (optimal PLA model)

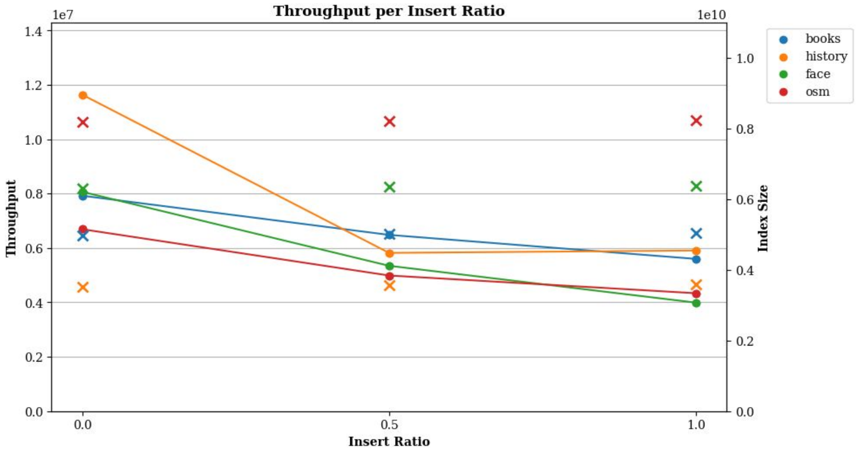
- But our approach is based on **conflict count**, so it is not compatible

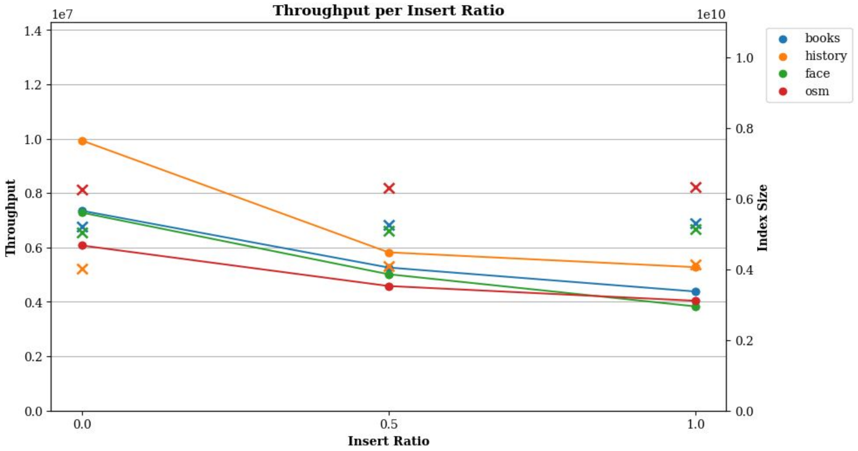
- Need to quantify to other criteria

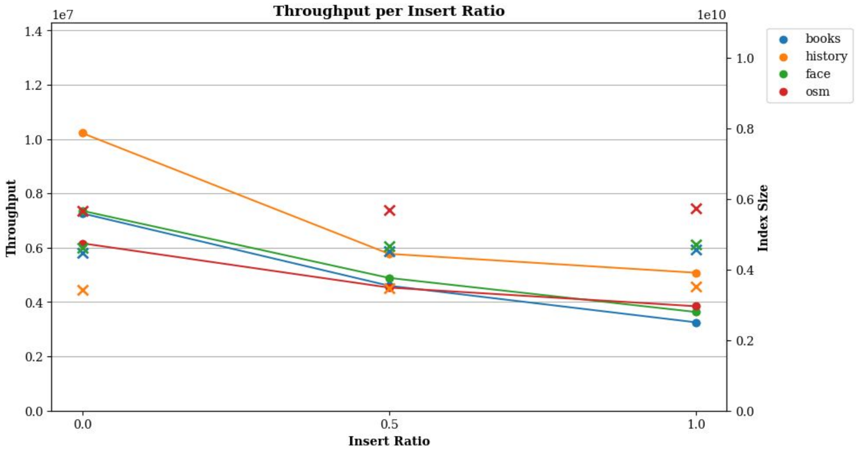
- Plan to use definition of conflict degree of node




Throughput per Insert Ratio
1e7
1e10
books
history
face
osm
Throughput
Index Size
Insert Ratio


Throughput per Insert Ratio
1e7
1e10
books
history
face
osm
Throughput
Index Size
Insert Ratio


Throughput per Insert Ratio
1e7
1e10
books
history
face
osm
Throughput
Index Size
Insert Ratio

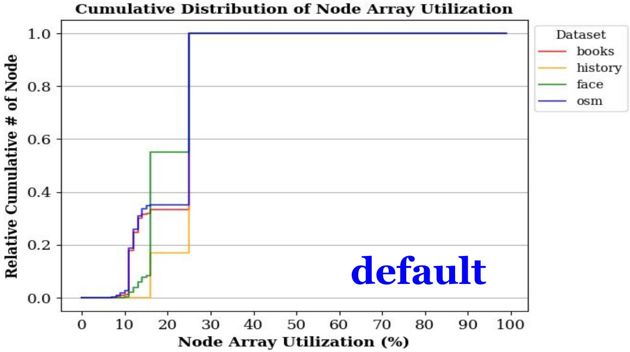

Throughput per Insert Ratio
1e7
1e10
books
history
face
osm
Throughput
Index Size
Insert Ratio

num_items: 8, 6, 4

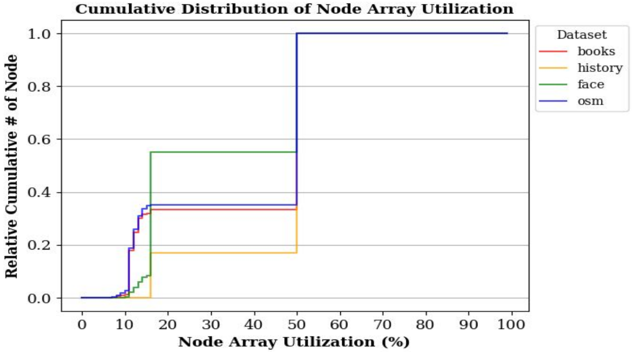
GAP_CNT: 1/2/5

GAP_CNT: $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{6}$

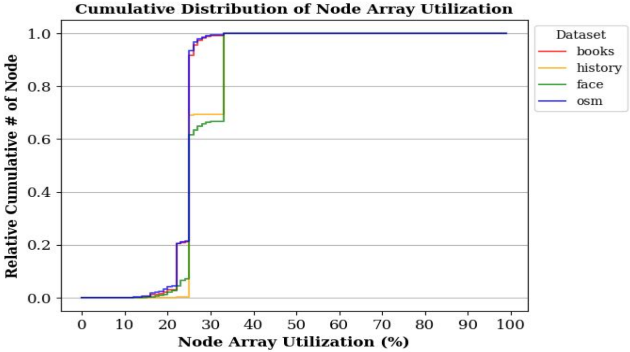
num_items: 8

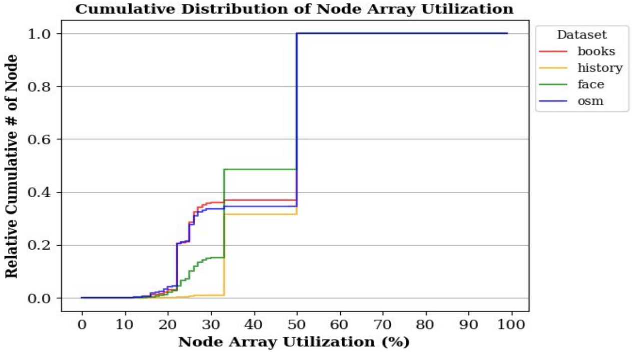


num_items: 4



GAP_CNT: 0/1/2

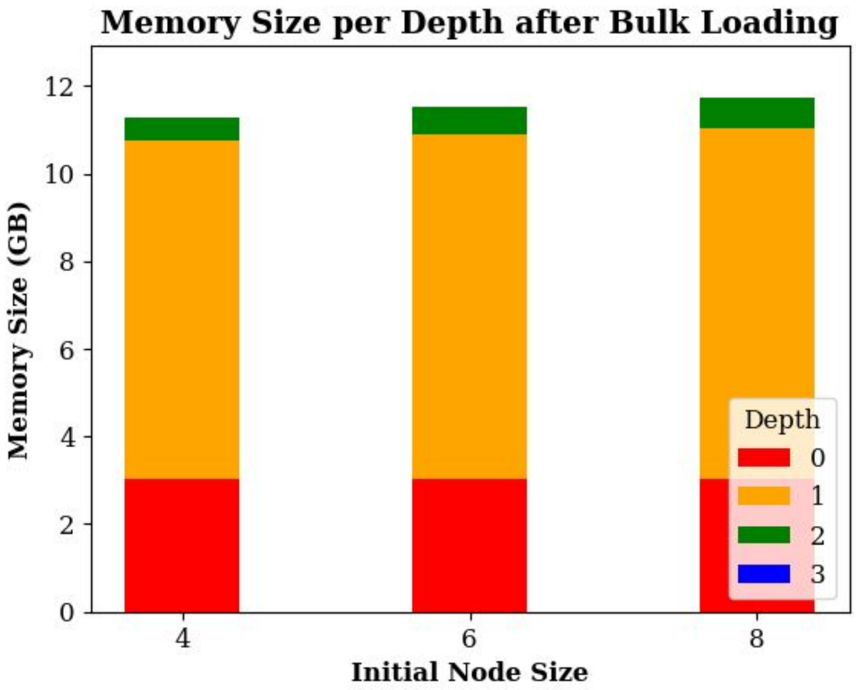


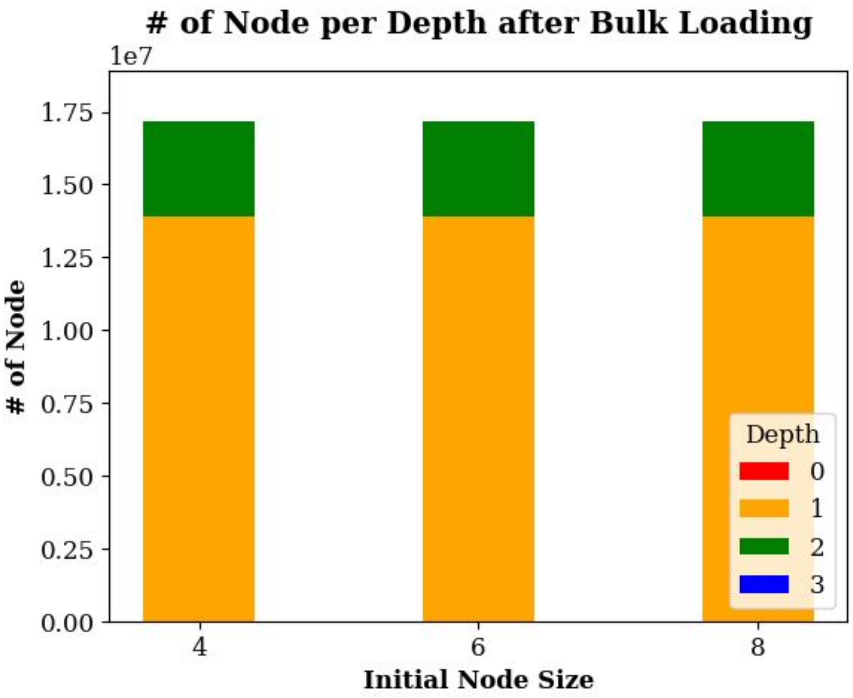


2. Observations

Observed after bulk load **100M** keys
Dataset: **Face**

3) Root Node Size (Initial Node Size)

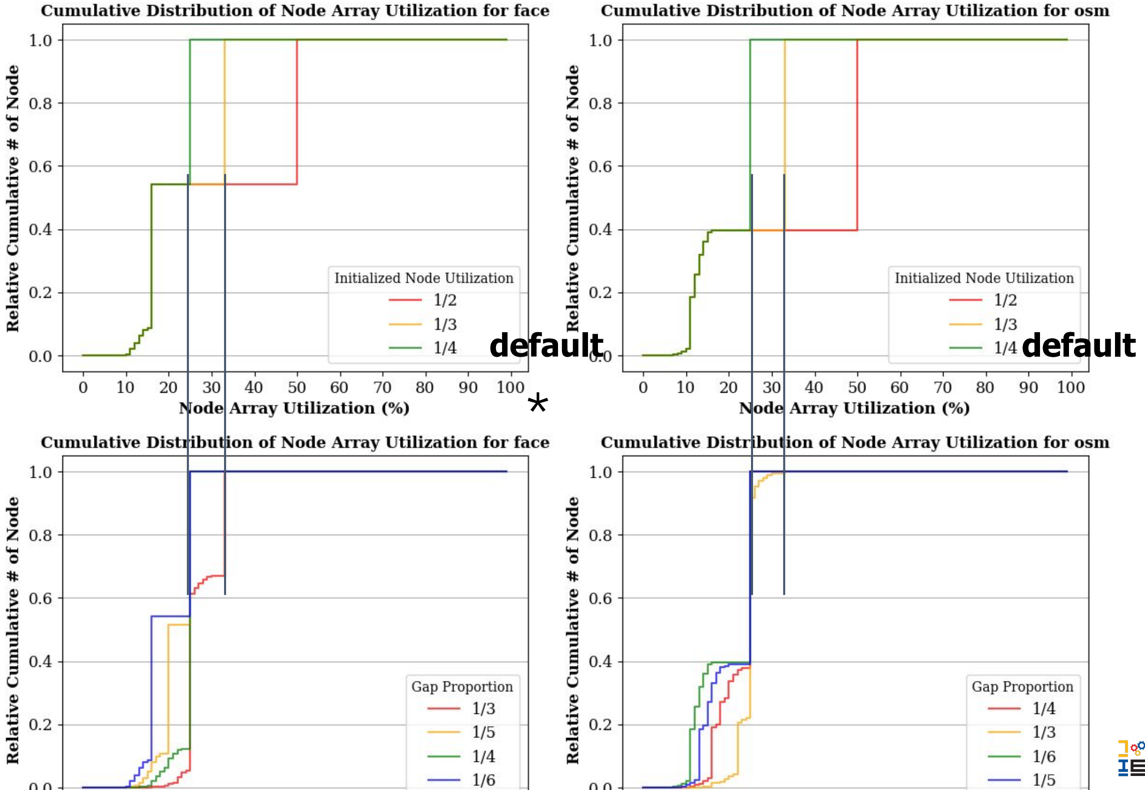




3. Experiment

Observed after bulk load **100M** keys

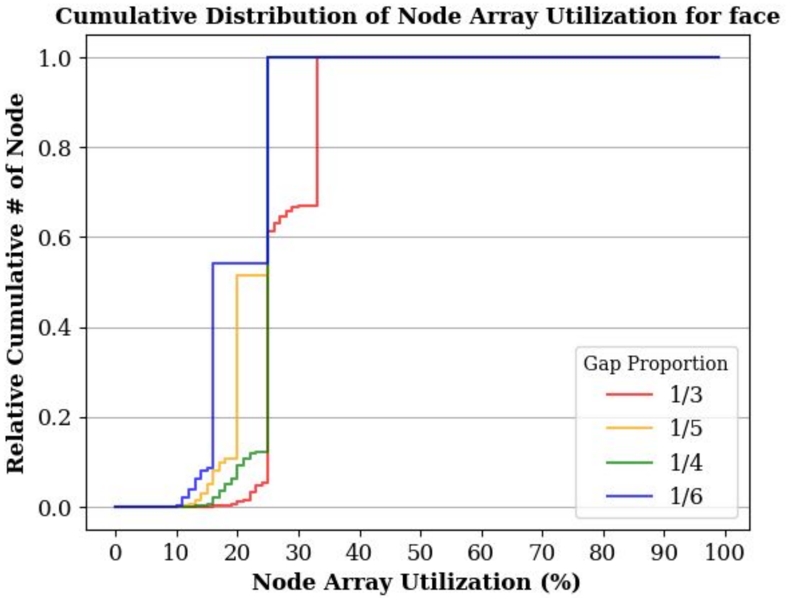
Array Utilization CDF (Initialized Node Size)

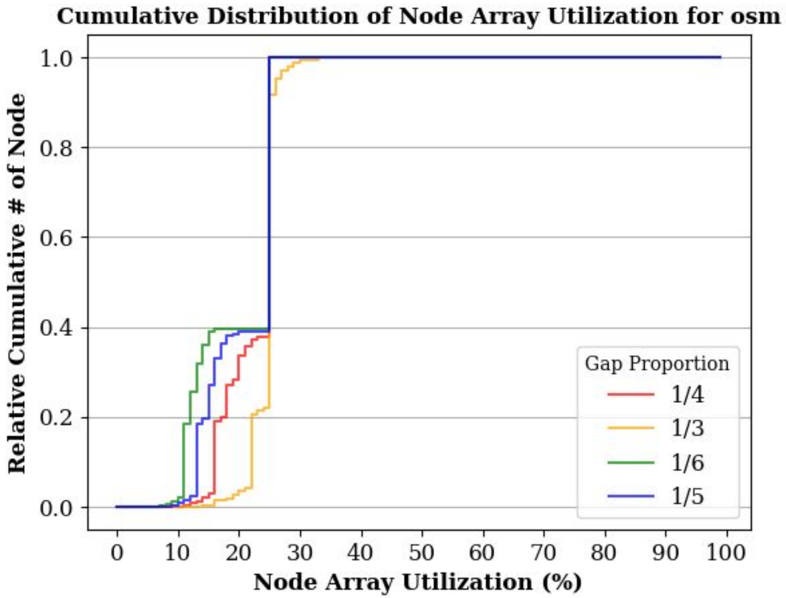


3. Experiment

Observed after bulk load **100M** keys

Array Utilization CDF (Gap Proportion)

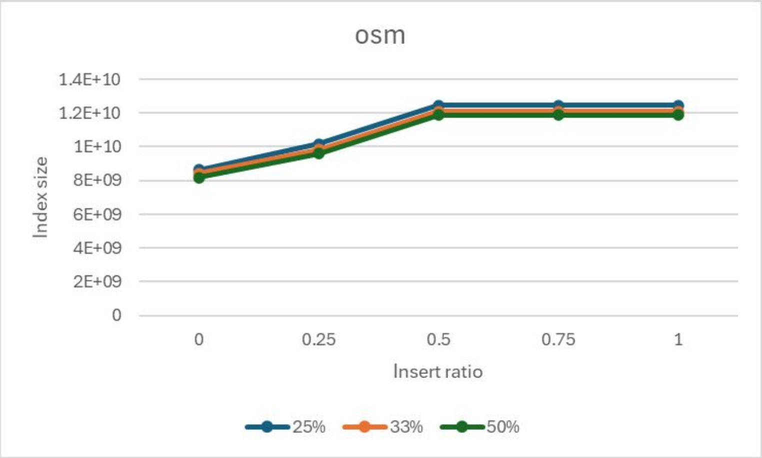


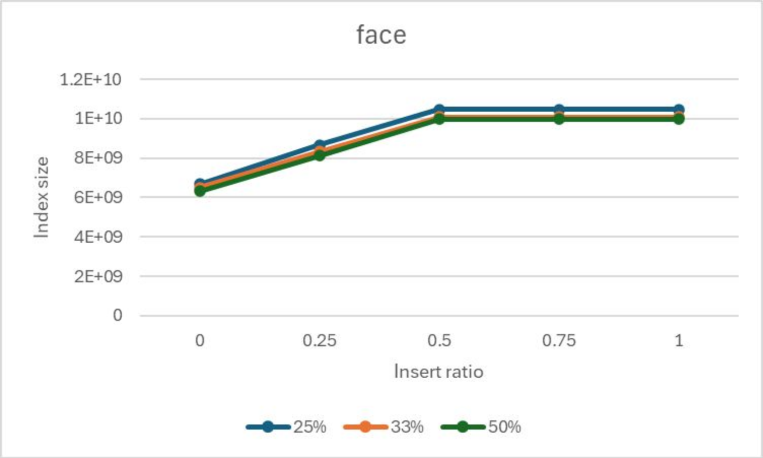


3. Experiment

Goal : lipp의 array utilization policy의 영향 (Size)

Observation . 우리는 개 삽질을 한것이다. 알파 베타가 답이었다.



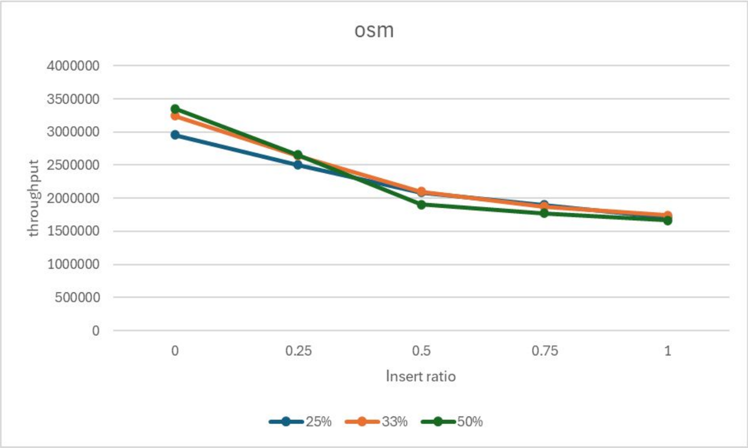


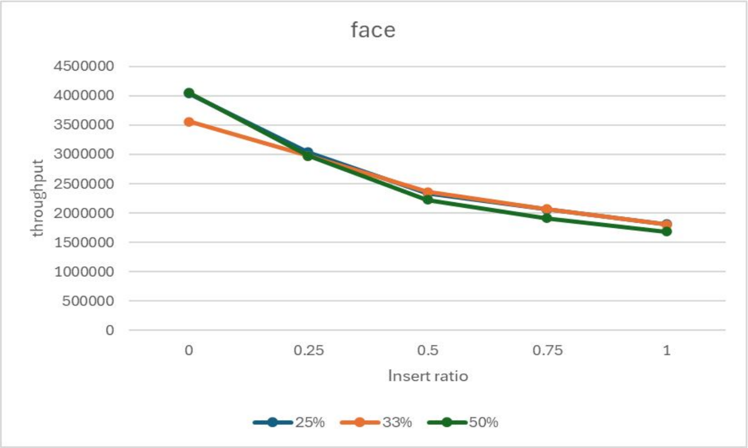
3. Experiment

Goal : lipp의 array utilization policy의 영향 (Performance)

read-write mixed workload

Observation . 모름

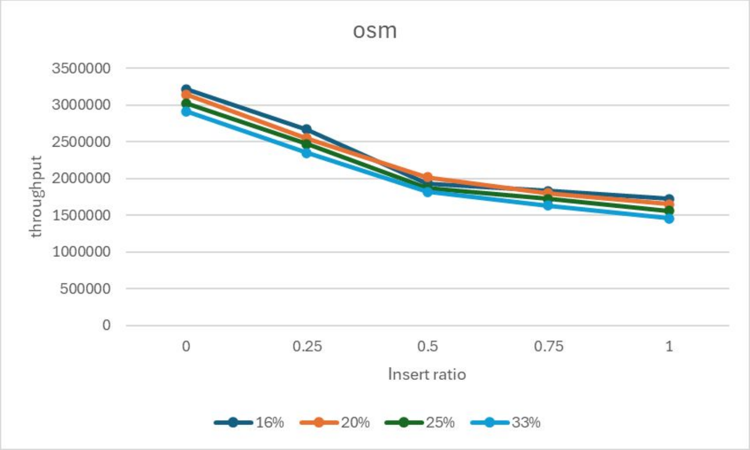


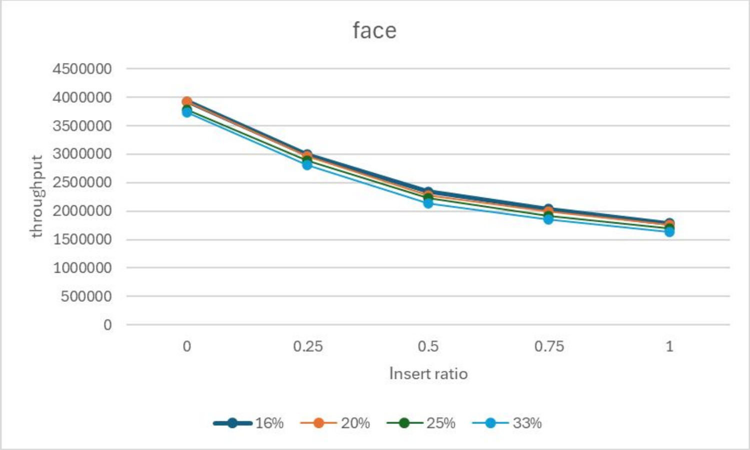


3. Experiment

Goal : lipp의 array utilization policy의 영향 (Performance)

Observation . 모름

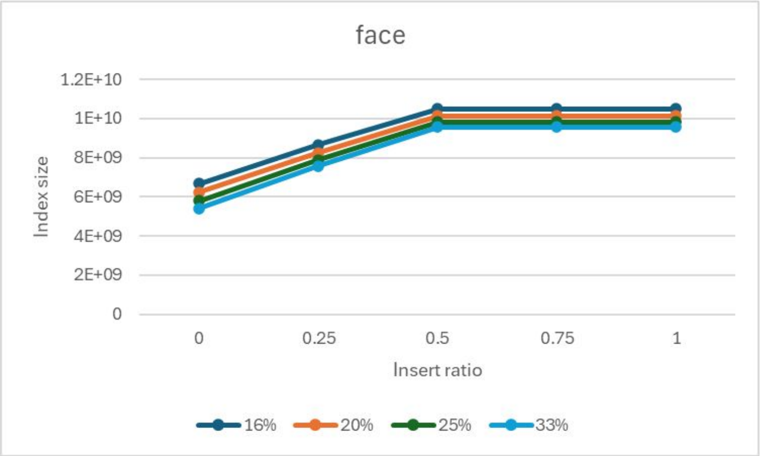




3. Experiment

Goal : lipp의 array utilization policy의 영향 (Size)

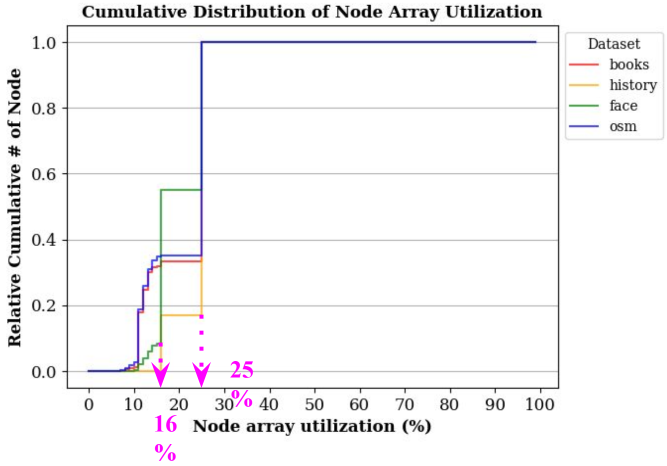
Observation . 모름



Hypothesis

Adjustment trigger

1) fill factor
2) conflict num



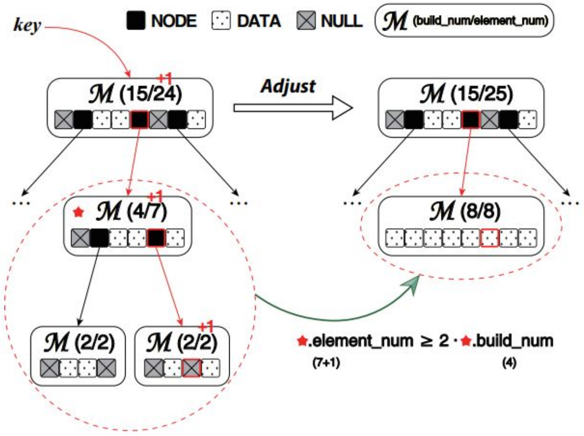


Figure 4: Node Adjustment

해당 노드가 아니라, sub-tree의 utilization을 고려하는 것이므로

beta가 2인 경우의 실제 utilization은 50% 보다 훨씬 작을 수 있음

좀 더 생각해보는 걸루

fill factor 가 문제면, 3/8도 존재해야함...

하지만 ⅜ 왜 없누? 누가 범인이누? 몰루?

아마도 FMCD, Conflict 문제인듯

좀 바꿔야 되긴 할듯요. 정리나 음 새로운 발견?
뭐든. 근데 제가 future work에 쓴거 보시면
가설을 2개로 나눠서 썼는데 보셨어요?

$$\frac{n.element\_num}{n.build\_num} \geq \beta \quad \beta \text{ is set to 2 by default}$$

$$\frac{n.conflict\_num}{n.element\_num - n.build\_num} \geq \alpha \quad \text{we set the threshold } \alpha = 0.1$$

4. Future work

1. A sensitivity analysis into updatable learned index structure
- **Array management policy : fill factor, initial node size (이 산이 아니다!)**
  - 사실 fill factor는 alpha, beta였던 거임 크크 (아님 말고 ㅋ)
- Model : simple linear regression vs kernelized linear regression (**FMCD**)
- Conflict resolving : shifting vs chaining
- SMO (Structural modification operation) : cost-benefit(fanout tree) vs rebuilding

2. Performance comparison between ALEX and LIPP through size
- Need understanding of fill factor parameter each indexes

3. Which techniques are appropriate when considering performance versus space?
- Create a new index based on that analysis
+ Conflict resolving : error-controlled approach (shift-chain hybrid)
+ Concurrency-friendly : semi-ordered