FITing-Tree: A Data-aware Index Structure[1]

Alex Galakatos* Michael Markovitch*
Carsten Binnig Rodrigo Fonseca Tim Kraska

Reported by

Zichen Zhu

Agenda

1. Introduction
2. Segmentation
3. Lookups & Inserts
4. Evaluation
5. Conclusion

Agenda

1 Introduction

2 Segmentation

3 Lookups & Inserts

4 Evaluation

5 Conclusion

1 Introduction – Motivation

Motivation 1 Existing index structures fail to exploit data patterns

Example: Internet of Things (IoT)

Data Types: √

- Time series
- Geospatial

Data Patterns: ✗

- Day/Night
- Class schedule
- Summer/Winter break
- Finals week

1 Introduction – Motivation

Motivation 2 Memory footprint is uncontrollable as data grows

	Tuples	Primary Indexes	Secondary Indexes
TPC-C	42.5%	33.5%	24.0%
Articles	64.8%	22.6%	12.6%
Voter	45.1%	54.9%	0%

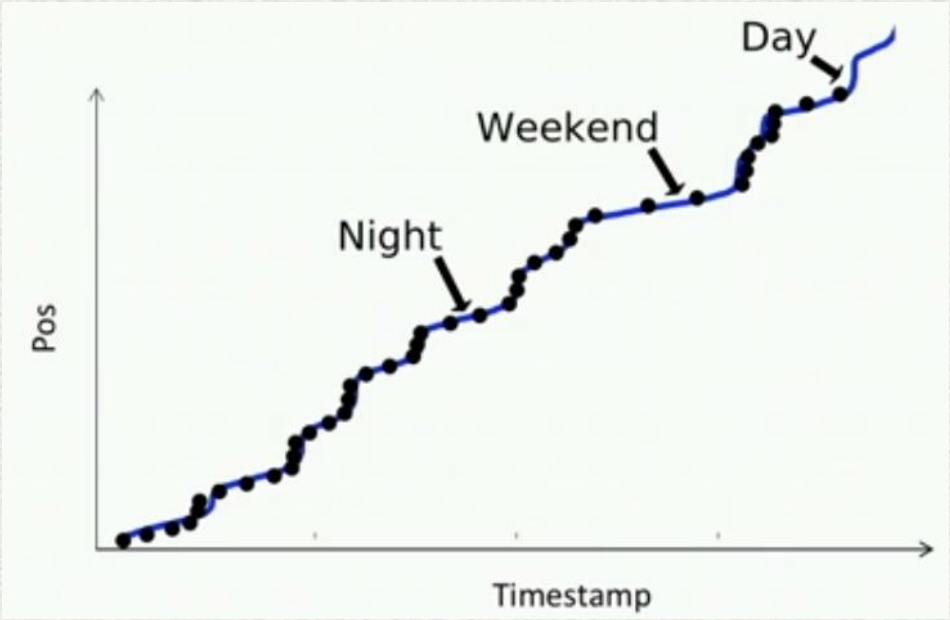
Percentage of the memory usage for tuples, primary indexes, and secondary indexes in **H-Store** using the default indexes (**B tree**) with DB size ≈10 GB[2]

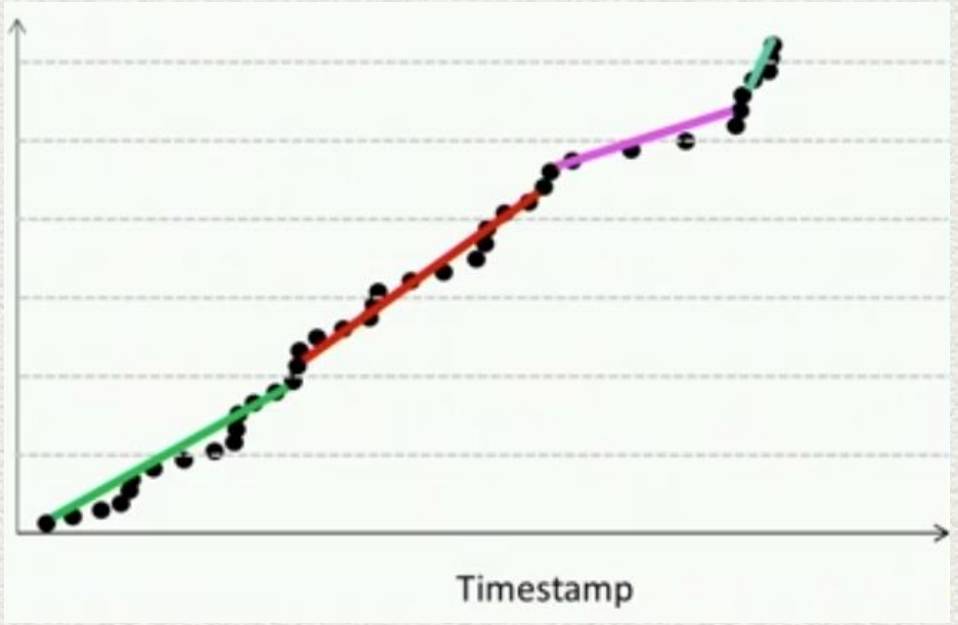
Performance: √
- Lookup
- Update

Storage Overhead: ×
- Budgetable
- Tradeoff navigation with performance

1 Introduction – Overview

Function Approximation

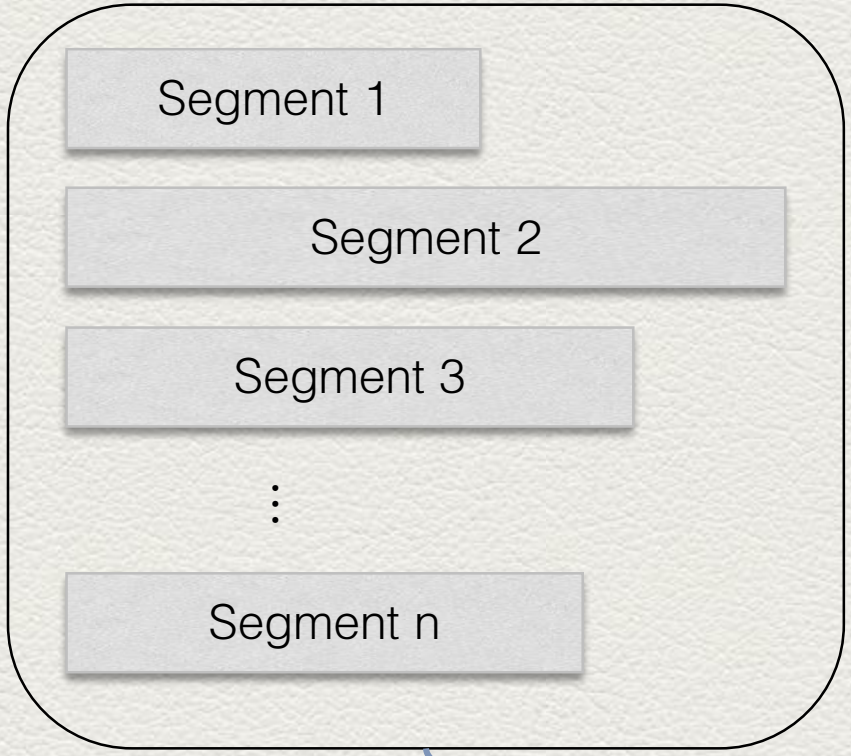


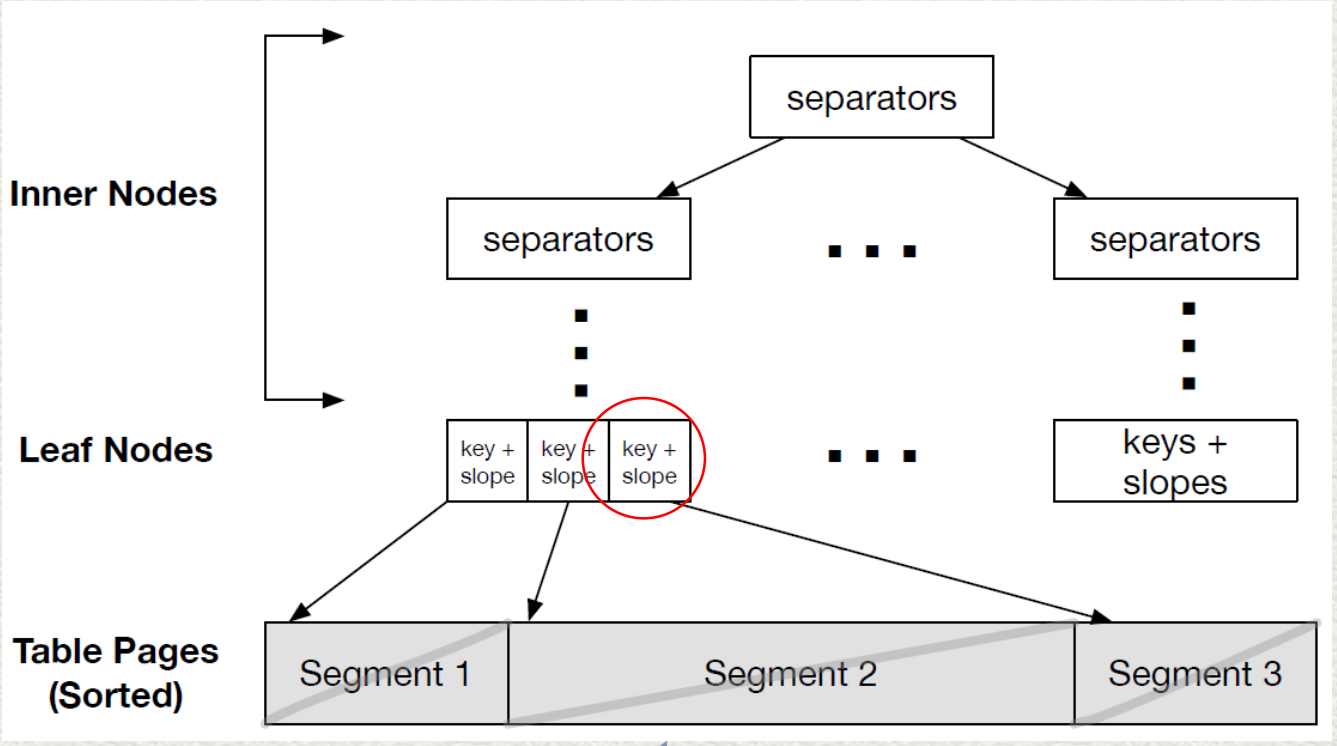


$$error = \max(|pred(k) - true(k)|) \forall k \in keys$$

1 Introduction – Overview

Clustered Segment Index





Agenda

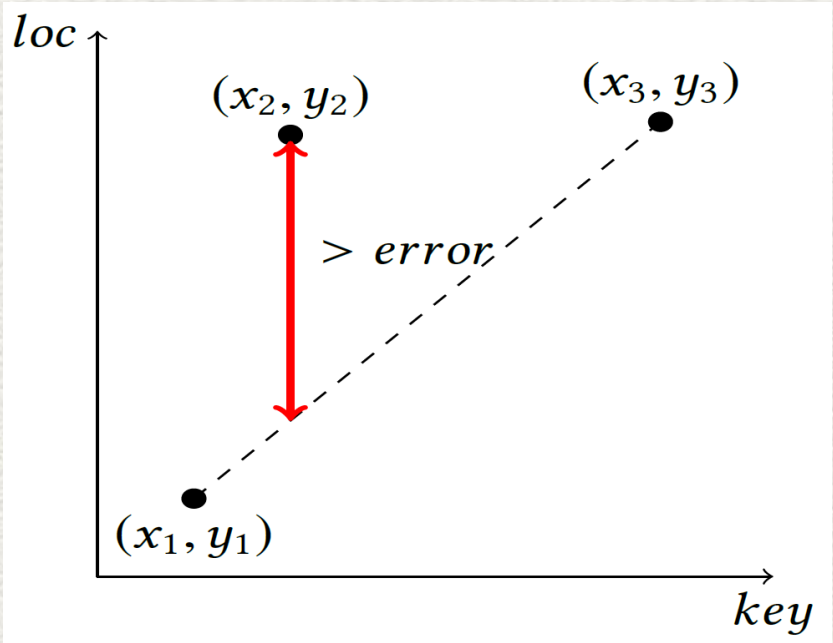
1. Introduction
2. Segmentation
3. Lookups & Inserts
4. Evaluation
5. Conclusion

2 Segmentation

Definition

A **segment** is a region of the key space that can be represented by a linear function whereby all keys are within a bounded distance (**error**) from their linearly interpolated position.

Example

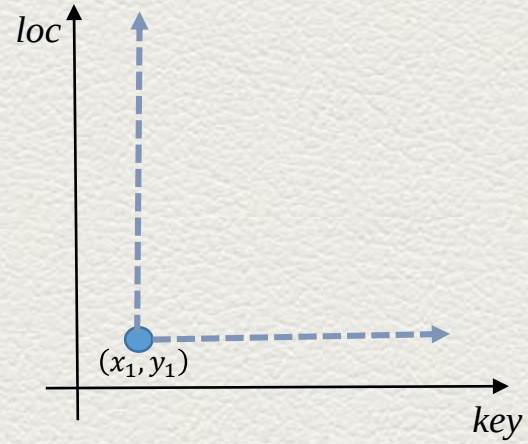


A **segment** from (x_1, y_1) to (x_3, y_3) is not valid if (x_2, y_2) is further than **error** from the interpolated line.

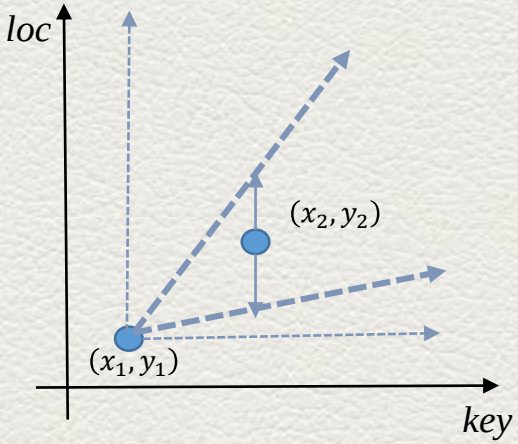
2 Segmentation

Shrinking Cone

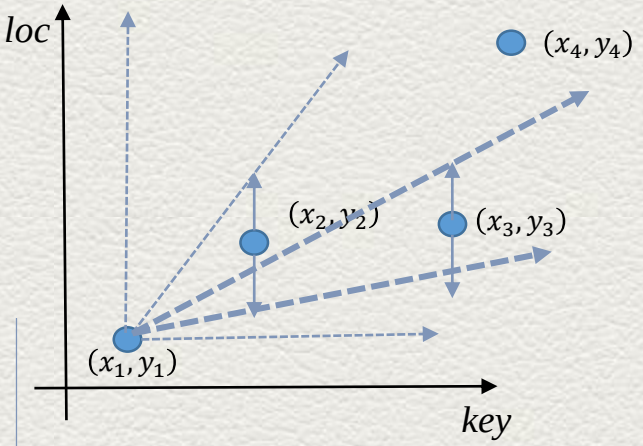
A **fast and efficient** algorithm but **not an optimal** one for segmentation



Starting with
a segment with
only one point (x_1, y_1)



A point (x_2, y_2)
is added



A point (x_3, y_3)
is added

What happens if
a point (x_4, y_4)
is added

Agenda

1 Introduction

2 Segmentation

3 Lookups & Inserts

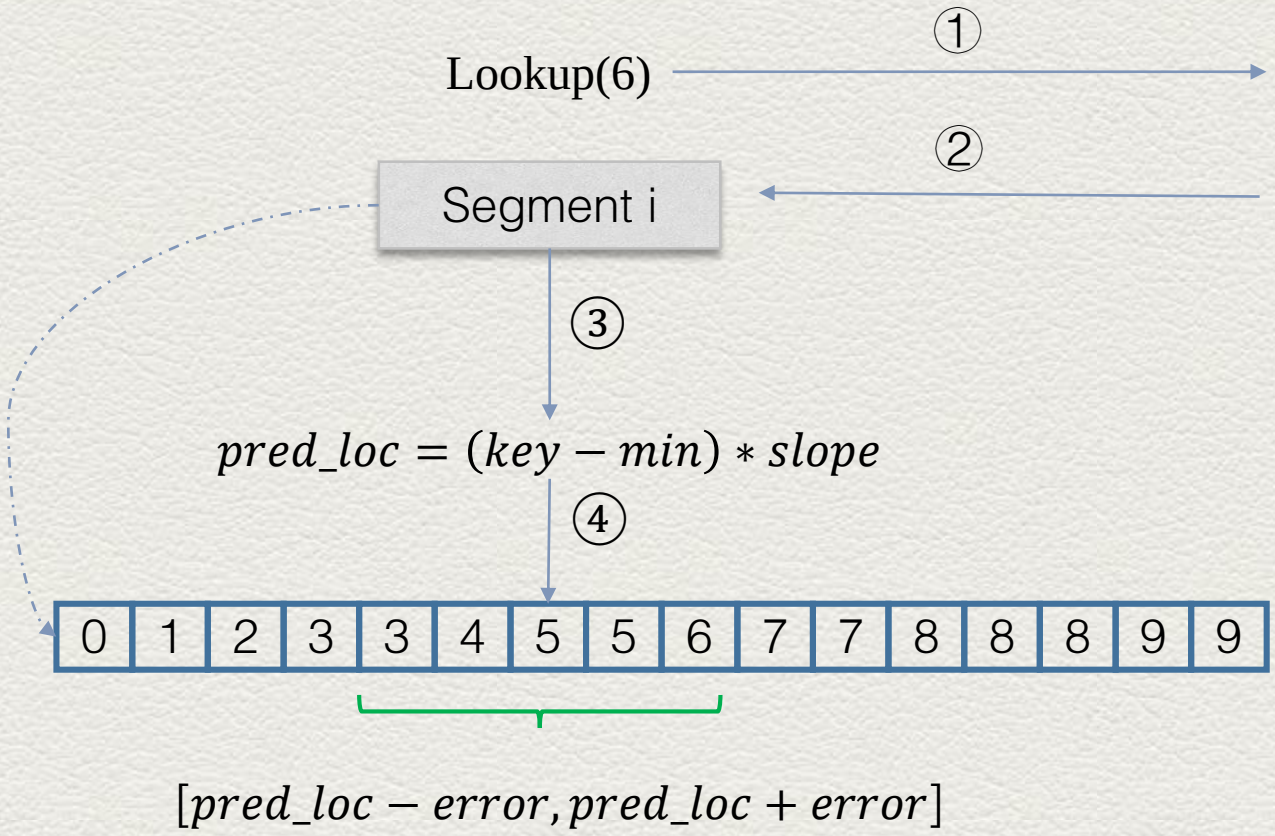
4 Evaluation

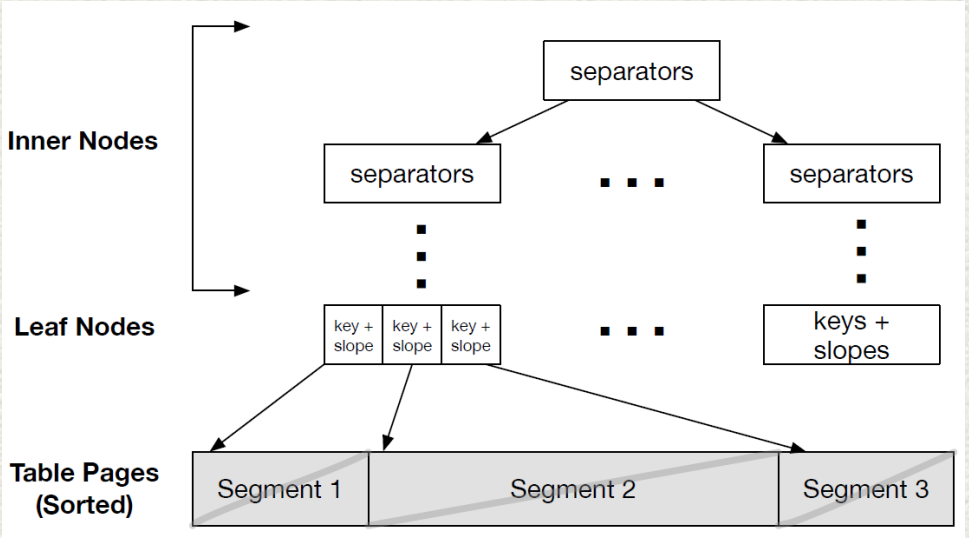
5 Conclusion

3 Lookups & Inserts

Lookups

Query the segment index and execute binary searching within a range

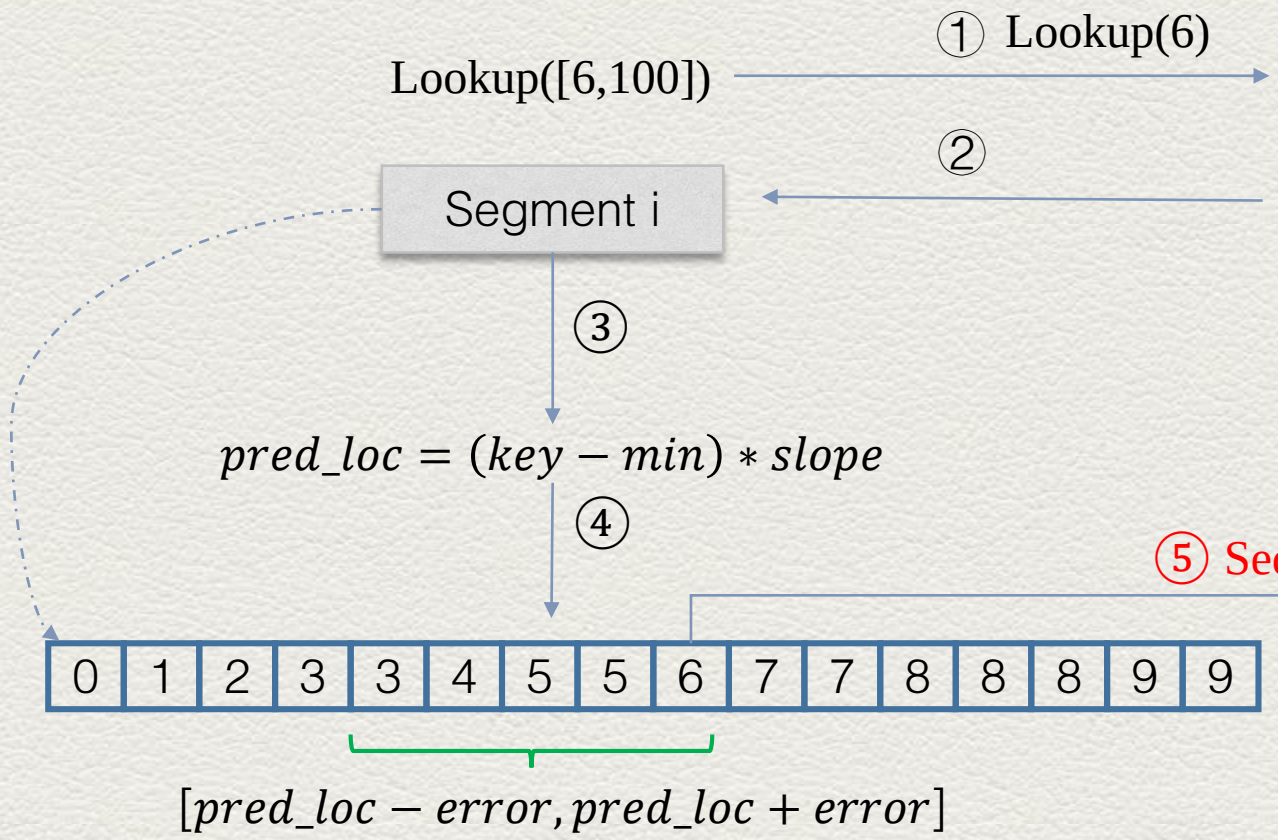


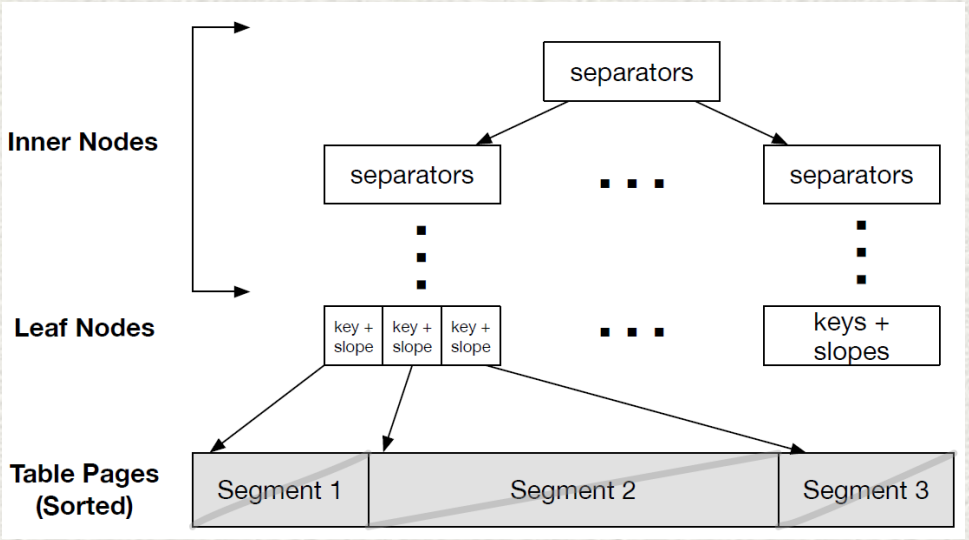


3 Lookups & Inserts

Lookups

Range query is executed by a point query of the starting key and the sequential scan





⑤ Sequential Scan until 100

Segment i+1 ··· Segment j

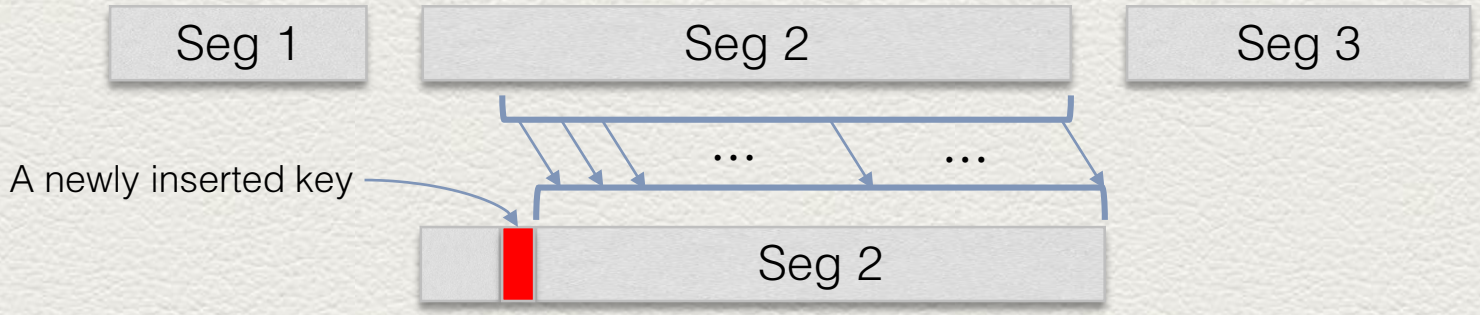
3 Lookups & Inserts

What is the problem of inserting in FITing-Tree?

Inserting in B+ Tree

- **In-place update** if the page is not full
- **In-place update** and split if the page is full

(1) Expensive for segment–based organization



3 Lookups & Inserts

What is the problem of inserting in FITing-Tree?

Inserting in B+ Tree

- **In-place update** if the page is not full
- **In-place update** and split if the page is full

(2) Is predicted error still bounded?

Lookup(6)

$pred\_loc = (key - min) * slope$

A newly inserted key

5

0	1	2	3	3	4	5	5	6	7	7	8	8	8	9	9

$loc(6) \in [pred\_loc - error, pred\_loc + error]$

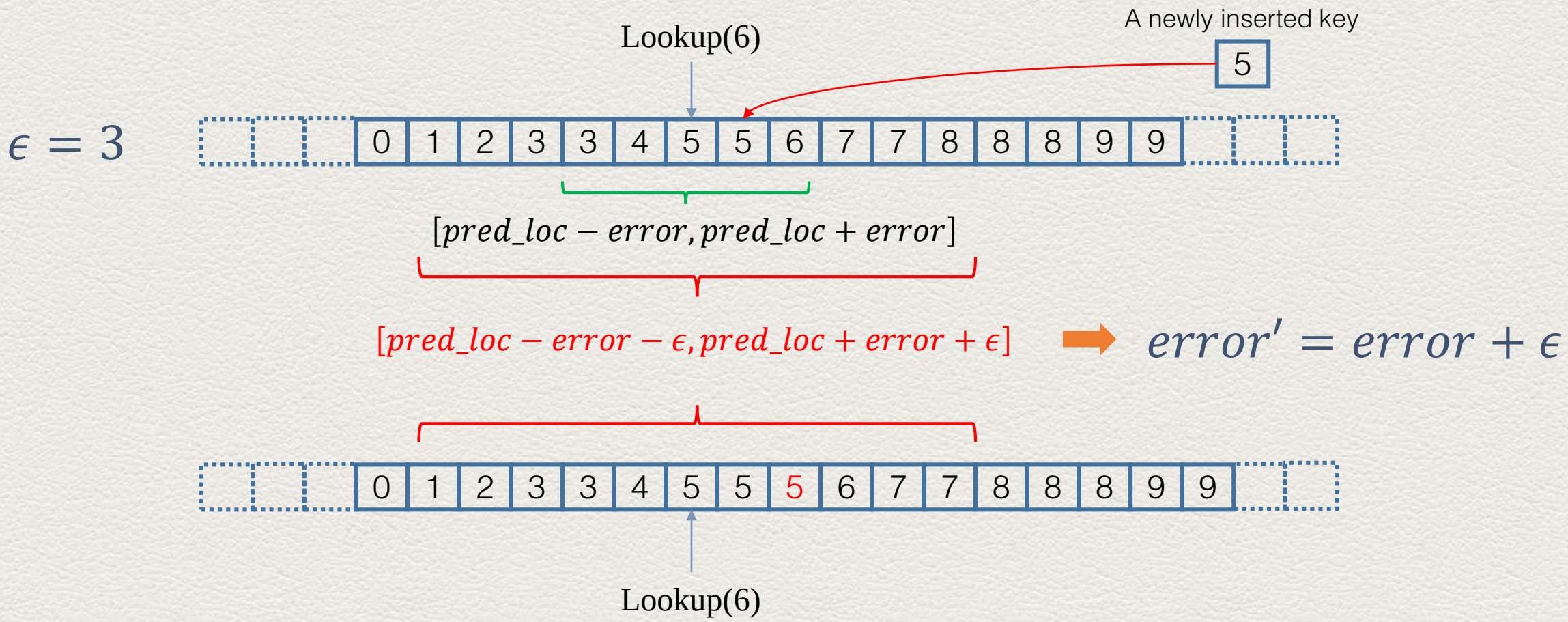
0	1	2	3	3	4	5	5	5	6	7	7	8	8	8	9	9

$loc(6) \notin [pred\_loc - error, pred\_loc + error]$

3 Lookups & Inserts

In-place Inserts

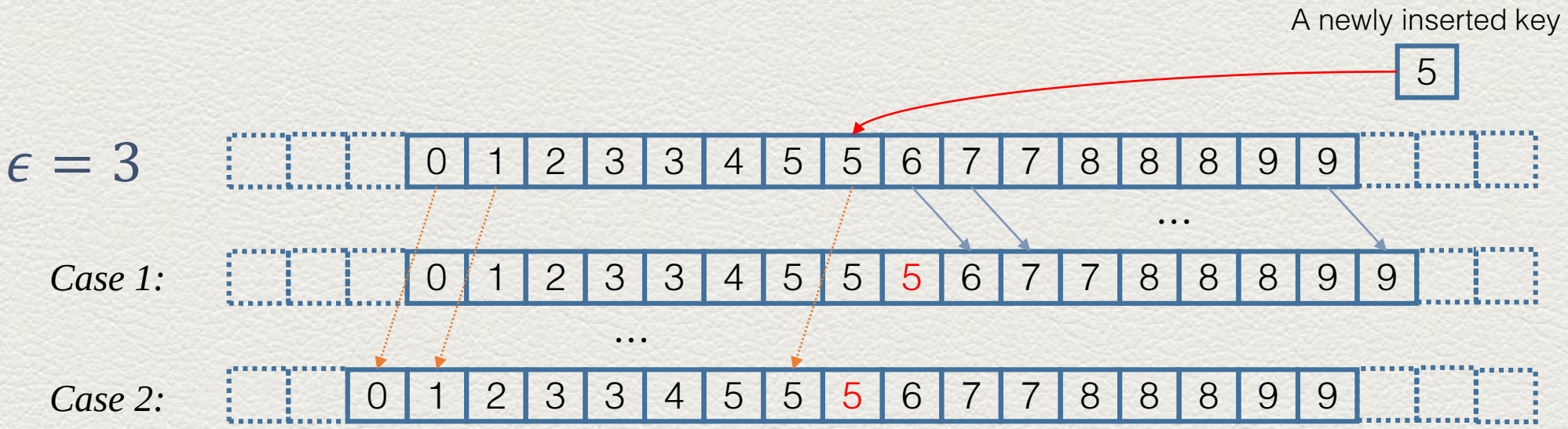
Leave 2ϵ free space for inserting and re-approximate segmentation once it is full



3 Lookups & Inserts

In-place Inserts

Bounded error is now maintained by ϵ but how about efficiency?



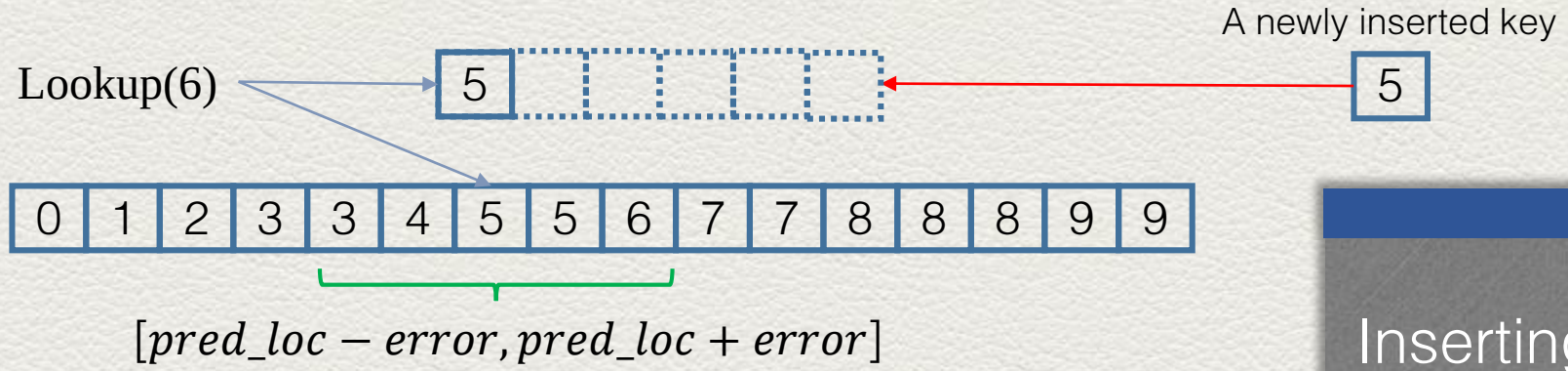
Choose the side with less element movement!

3 Lookups & Inserts

Delta Inserts

Leave ϵ free space for buffer and once it is full, merge with the segment and re-approximate segmentation

$\epsilon = 6$

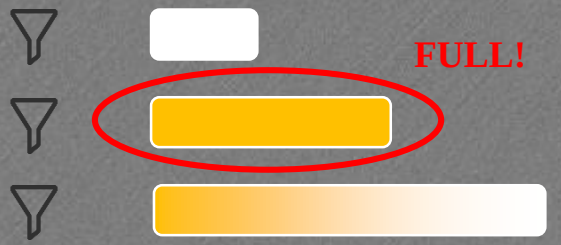


$error' = error + \epsilon$

$$\boldsymbol{Latency = c[\log_b(|S|) + \log_2(error) + \log_2(\epsilon)]}$$

$$\boldsymbol{Size = |S|\log_b(|S|) \cdot 16B + |S| \cdot (\epsilon + 24B)}$$

Inserting in LSM–Tree



Merge with the next level and re-program bloom filter to maintain lookup efficiency

Agenda

1 Introduction

2 Segmentation

3 Lookups & Inserts

4 Evaluation

5 Conclusion

4 Evaluation

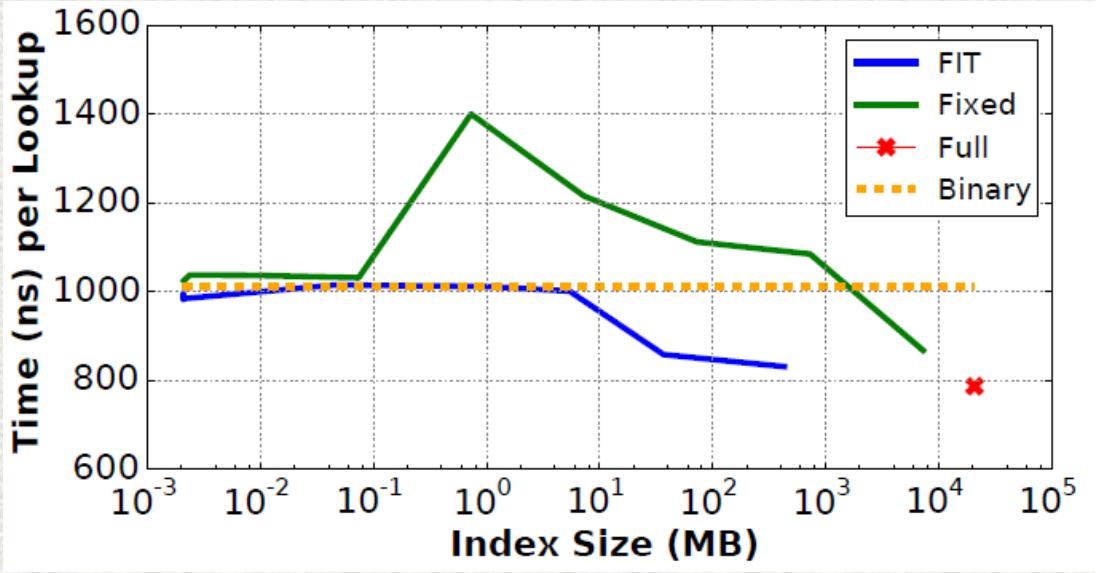
Dataset is Important!

	Size	Index	Patterns
Weblogs	715M	Clustered Index	Day/Night
IoT	5M	Clustered Index	Class schedule
Maps	2B	Non-clustered Index	-

4 Evaluation

(1) Lookup Latency vs Index Size

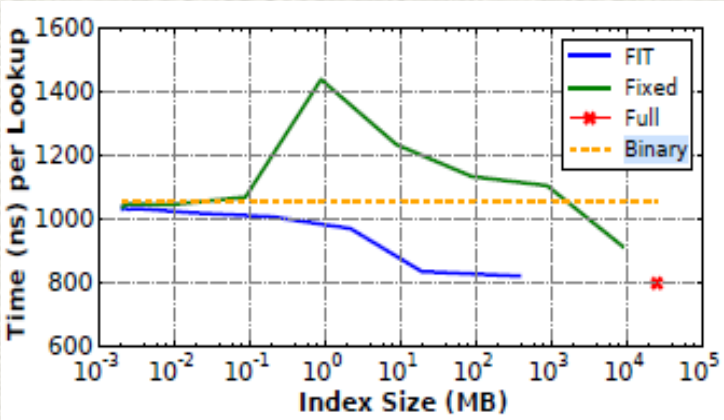
- STX-tree (B^+ tree implementation)
- Baselines
  - Full index (a dense index, there is an index pointer for each data record)
  - Fixed-size paging (a sparse index, index records are not created for every key)
  - Binary search



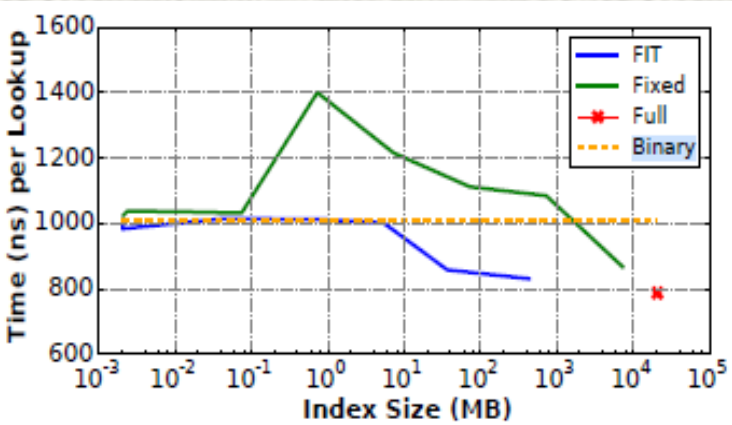
IoT Dataset

4 Findings

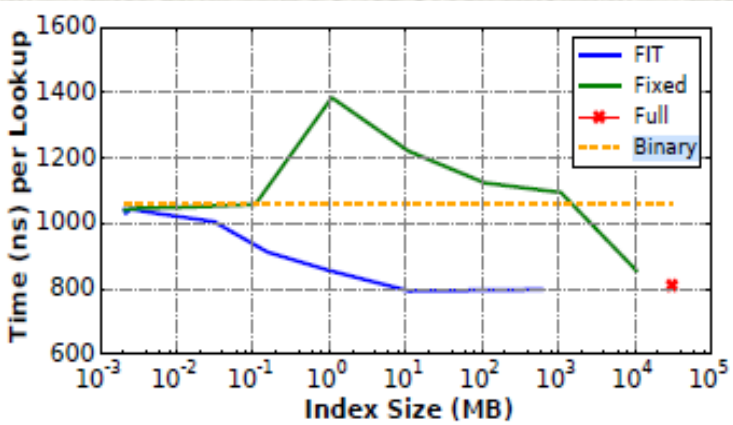
(1) Lookup Latency vs Index Size



(a) Weblogs



(b) IoT

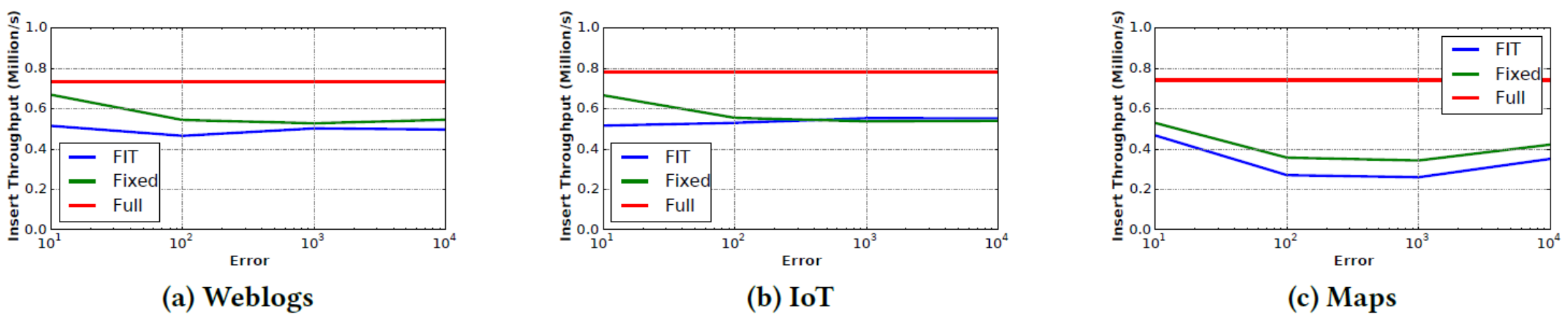


(c) Maps

FITing-Tree offers very low lookup latency with significant space saving

4 Findings

(2) Throughput for Inserts vs Error

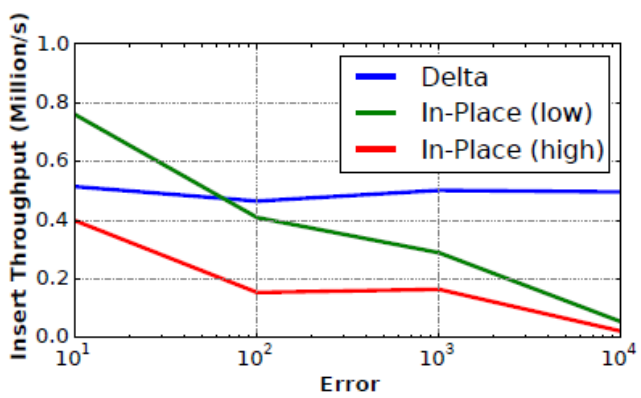


(a) Weblogs (b) IoT (c) Maps

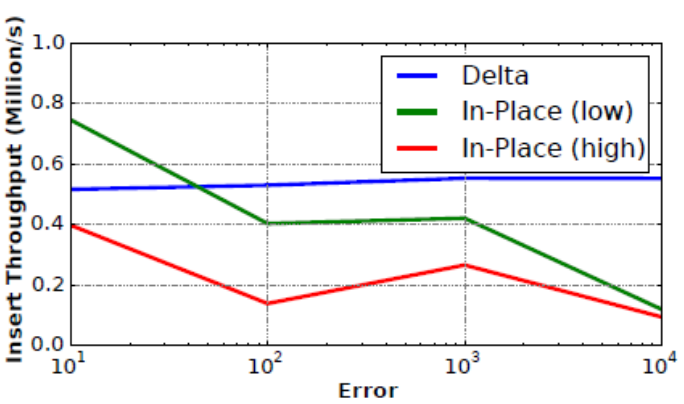
FIT does not provide the highest write throughput due to extra cost on segmentation

4 Findings

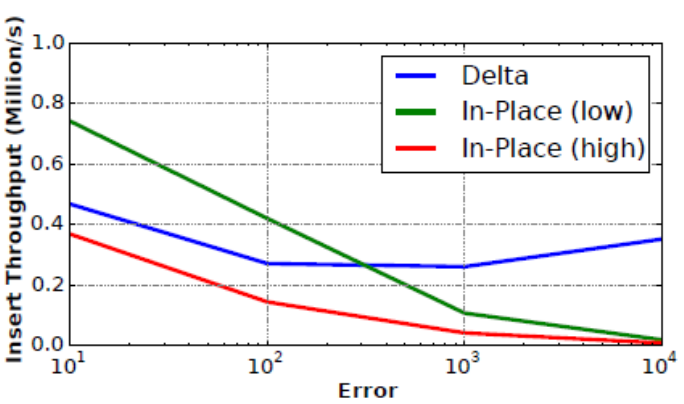
(3) Insertion Strategy Microbenchmark



(a) Weblogs



(b) IoT



(c) Maps

In-place strategy with a low fill factor offers the highest insert performance

Agenda

1. Introduction
2. Segmentation
3. Lookups & Inserts
4. Evaluation
5. Conclusion

5 Conclusion

- FITing-Tree uses **piece-wise linear functions** to approximate the distribution to support efficient lookup

- FITing-Tree presents an index that introduces **a tunable parameter ϵ to balance the tradeoff** between lookup performance and space consumption of an index

- The segment-based structure can be easily **integrated with many existing index** structures (e.g. B^+ tree and FAST) and thus has potential application prospects

5 Conclusion

Can we do better?

- Delta-insert strategy can allow buffer is unsorted to improve the write efficiency in OLTP workload
- A few meta data can be added to support efficient aggregate query such as MAX/SUM query
- Segmentation algorithm should consider the indexing structure to derive a more suitable segmentation scheme.

Citations

[1] Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R. and Kraska, T., 2019, June. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data* (pp. 1189-1206).

[2] Zhang, H., Andersen, D.G., Pavlo, A., Kaminsky, M., Ma, L. and Shen, R., 2016, June. Reducing the storage overhead of main-memory OLTP databases with hybrid indexes. In *Proceedings of the 2016 International Conference on Management of Data* (pp. 1567-1581).

Thanks

Reported by

Zichen Zhu