Indexing for Near-Sorted Data

Aneesh Raman
Matthaios Olma

Subhadeep Sarkar
Manos Athanassouli
Indexes in Databases

- organize data
- efficient queries

The process of inducing "sortedness" to an otherwise unsorted data collection
What if data already has some structure?
What if data already has some structure?

Near-sorted data
What if data already has some structure?

Near-sorted data

\[ \approx \]

treated same as unstructured data!
Intermediate-Sortedness in Practice

- Time Series
- Stock market
- Join/query

Efficient reads + fast writes

Classical indexes carry redundant effort!
Ideally...

Standard ingestion

Pay less cost as data becomes more sorted

Ideal ingestion

Ingestion cost

Bulk loading

Scrambled

Increasing data sortedness

Sorted
The Sortedness-Aware (SWARE) Paradigm
Sortedness-Aware (SWARE) Paradigm

intelligent buffering + opportunistic bulk loading + increased fill and split factor

SWARE framework can be applied to any tree-index!
SWARE Ingestions

Buffer

Zonemap
(min-max)

flush 3 pages

SWARE Buffer

non overlapping pages may move

flush non-overlapping pages to tree

B+-tree

Leaf pages

tail leaf node

non-overlapping pages

...
**SWARE Ingestions**

- **Buffer**
- **Zonemap (min-max)**
- **B+-tree**
- **Leaf pages**

- **Non-overlapping pages**: may move
- **Flush non-overlapping pages to tree**
- **Bulk load page-by-page if in order**

- **Flush 3 pages**

- **Tail leaf node**

---

This diagram illustrates the SWARE Ingestions system, focusing on buffer management, zonemap, and B+-tree operations.
SWARE Ingestions

- Buffer
- Zonemap (min-max)

- tail leaf node
- non-overlapping pages

B+-tree

- non overlapping pages may move
- flush non-overlapping pages to tree
- bulk load page-by-page if in order

Leaf pages
SWARE Ingestions

Buffer

Zonemap (min-max)

Buffer

SWARE Buffer

move & sort remaining entries

B+-tree

Leaf pages

non-overlapping pages may move

flush non-overlapping pages to tree

bulk load page-by-page if in order

tail leaf node

non-overlapping pages
SWARE Ingestions

Buffer
Zonemap (min-max)

(update non-overlapping pages)

SWARE Buffer

- tail leaf node
- non-overlapping pages

B+-tree

Leaf pages

- non overlapping pages may move
- flush non-overlapping pages to tree
- bulk load page-by-page if in order

BOSTON UNIVERSITY
SWARE Ingestions

Buffer
Zonemap (min-max)

850-910
915-970
974-1030
1032-1088
1090-1500

SWARE Buffer

B+-tree

- tail leaf node
- non-overlapping pages
- fully sorted pages

Leaf pages

non overlapping pages may move
flush non-overlapping pages to tree
bulk load page-by-page if in order
How do lookups work?
Overall Structure for Queries

- **Global Bloom filter**
  - Buffer
  - Zonemap (min-max)
  - Per-page Bloom filters

- **B+-tree**
  - Sorted section uses faster interpolation search

- **SWARE Buffer**
  - Global BF. helps skip buffer probe
  - Per-page BFs eliminate page-scans

- **Leaf pages**
  - Tail leaf node
  - Non-overlapping pages
  - Fully sorted pages
How do we evaluate SWARE?
Benchmark on Data Sortedness (TPCTC 2022)

#. unordered entries = K

[BenMoshe, ICDT 2011]

max. displacement among unordered entries = L

Insert Only

Mixed Workloads
( interleaved reads and writes)
Experimental Setup

Metrics:
1. Overall performance (speedup)
2. Raw performance (latency)

Workload Generator: BoDS
1. 500M Integer keys (~ 4GB)
2. Random existing lookups

System Setup:
1. Intel Xeon Gold 5230
2. 2.1GHz processor w. 20 cores
3. 384GB RAM, 28MB L3 cache

Default Index Setup:
1. Buffer = 40MB; flush <= 50%
2. BFs = 10 BPK; Murmur Hash
3. Split at 80%
Overall Performance

≅ 9x speedup

Overall Speedup:

- Fully sorted
- Near-sorted
- Less sorted
- Scrambled
- B⁺-tree Cost

Read:Write Ratio: 10:90 25:75 40:60 50:50 60:40 75:25 90:10

Speedup:

- K=0%
- K=5%, L=5%
- K=50%, L=50%

Overall Performance:

K=0%
Overall Performance

\[ \approx 9 \times \text{speedup} \]

\[ \approx 4 \times \text{for mixed reads & writes} \]
Overall Performance

The overall performance is comparable to a B+-tree with less sortedness.
Raw Ingestion Performance

Ingestion latency reduced between 27-90%
Raw Ingestion Performance

**Ingestion latency reduced between 27-90%**

**Bulk loading is maximized with high data sortedness**
## Space Efficiency

<table>
<thead>
<tr>
<th>Sortedness Degree</th>
<th>B+ tree</th>
<th>SA B+ tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Sorted</td>
<td>2.004M (8K, 1.996M)</td>
<td>0.52x</td>
</tr>
<tr>
<td>Near-Sorted</td>
<td>1.847M (7K, 1.840M)</td>
<td>0.6x</td>
</tr>
<tr>
<td>Less-Sorted</td>
<td>1.878M (4.3K, 1.873M)</td>
<td>1.01x</td>
</tr>
</tbody>
</table>
## Space Efficiency

<table>
<thead>
<tr>
<th>Sortedness Degree</th>
<th>#. Nodes (#. Internal, #. Leaf)</th>
<th>B+ tree</th>
<th>SA B+ tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Sorted</td>
<td>2.004M (8K, 1.996M)</td>
<td></td>
<td>0.52x</td>
</tr>
<tr>
<td>Near-Sorted</td>
<td>1.847M (7K, 1.840M)</td>
<td></td>
<td>0.6x</td>
</tr>
<tr>
<td>Less-Sorted</td>
<td>1.878M (4.3K, 1.873M)</td>
<td></td>
<td>1.01x</td>
</tr>
</tbody>
</table>

*increased fill/split factor helps reduce memory footprint*
Summary

Identify “sortedness” as a resource

Smart buffering + bulk index appends = faster ingestion

Works well with write-heavy or mixed read-write workloads

Thank You!

✉️: aneeshr@bu.edu
🌐: ramananeesh.com/