Class 27: Log-Structured-Merge Trees

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/

based on slides from Niv Dayan
Useful when?

- Massive dataset
- Rapid updates/insertions
- Fast lookups

⇒ LSM-trees are for you.
Why now?

Patrick O'Neil
UMass Boston

Invented in 1996

Time
1980 1990 2000 2010
Outline

1. Storage devices
2. Indexing problem & basic solutions
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters
Storage devices
The Memory Hierarchy

- Metadata & frequently accessed data
- Main Memory (expensive, fast)
- All data
- Disk (cheap, slow)
≈ 100 ns

≈ 10 ms

≈ 5-6 order of magnitude difference
Why is disk slow?

Random access is slow $\implies$ move disk head

Sequential access is faster $\implies$ let disk spin
64 byte chunks
Words

4 kilobyte chunks
Blocks

Fine access granularity

Coarse access granularity
64 byte chunks
Words

4 kilobyte chunks
Blocks

Fine access granularity

Coarse access granularity
Outline

1. Storage devices
2. Indexing problem & basic solutions
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters
Outline

1. Storage devices
2. Indexing problem & basic solutions
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters
Indexing Problem & Basic Solutions
Indexing Problem

names  \rightarrow  phone numbers

Structure on disk?
Lookup cost?
Insertion cost?
## Results Catalogue

Compare and contrast data structures.
What to use when?

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results Catalogue

Compare and contrast data structures.
What to use when?

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Modeling Performance

- 64 byte Words: \(\approx 1\) ns
- 4 kilobyte Blocks: \(\approx 100\) ns

Measure bottleneck:
Number of block reads/writes (I/O)
Sorted Array

N entries
B entries fit into a disk block
Array spans N/B disk blocks

Lookup method & cost?

Binary search: \( O \left( \log_2 \left( \frac{N}{B} \right) \right) \) I/Os

Insertion cost?

Push entries: \( O \left( \frac{1}{B} \cdot \frac{N}{B} \right) \) I/Os

Buffer
James
Sara

Array size
Pointer

Block 1 | Block 2 | ... | Block N/B
--- | --- | --- | ---
Anne | Bob | ... | Yulia
Arnold | Corrie | ... | Zack
Barbara | Doug | ... | Zelda
# Results Catalogue

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B^2)$</td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Results Catalogue

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B^2)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Log  (append-only array)

\( \text{N entries} \)
\( \text{B entries fit into a disk block} \)
Array spans \( \frac{\text{N}}{\text{B}} \) disk blocks

Lookup method & cost?
Scan: \( O\left(\frac{\text{N}}{\text{B}}\right)\)
Insertion cost?
Append: \( O\left(\frac{1}{\text{B}}\right)\)

Buffer
- James
- Sara

Array size  Pointer

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block ( \frac{\text{N}}{\text{B}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doug</td>
<td>Yulia</td>
<td></td>
<td>Anne</td>
</tr>
<tr>
<td>Zelda</td>
<td>Zack</td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>Arnold</td>
<td>Barbara</td>
<td></td>
<td>Corrie</td>
</tr>
</tbody>
</table>
## Results Catalogue

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B^2)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B^2)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B-tree

Lookup method & cost?
Tree search: $O \left( \log_B \left( \frac{N}{B} \right) \right)$

Insertion method & cost?
Tree search & append: $O \left( \log_B \left( \frac{N}{B} \right) \right)$

Depth: $O(\log_B(N/B))$
# Results Catalogue

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B^2)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B-trees

“It could be said that the world’s information is at our fingertips because of B-trees”

Goetz Graefe Microsoft, HP Fellow, now Google ACM Software System Award
B-trees are no longer sufficient

Cheaper to store data

Workloads more insert-intensive

We need better insert-performance.
# Results Catalogue

Goal to combine

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sorted array</strong></td>
<td>$\mathcal{O}(\log_2(N/B))$</td>
<td>$\mathcal{O}(N/B^2)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$\mathcal{O}(N/B)$</td>
<td>$\mathcal{O}(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$\mathcal{O}(\log_B(N/B))$</td>
<td>$\mathcal{O}(\log_B(N/B))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Leveled LSM-tree</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tiered LSM-tree</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic LSM-trees
Basic LSM-tree

Buffer

Sorted arrays

Level

0

1

2

3
Basic LSM-tree

Design principle #1: optimize for insertions by buffering
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

- **Buffer**
  - Level 0
  - Level 1
  - Level 2
  - Level 3

- **Sorted arrays**

- **Inserts**
  - sort & flush buffer
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

Buffer

Sorted arrays

Level

0

1

2

3

Inserts

sort & flush buffer
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

*Design principle #2:* optimize for lookups by sort-merging arrays

- **Buffer**
  - Level 0
  - Level 1
  - Level 2
  - Level 3

- **Sorted arrays**

**Inserts**

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

(sort & flush buffer)

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

*Design principle #2:* optimize for lookups by sort-merging arrays
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

*Design principle #2:* optimize for lookups by sort-merging arrays

**Inserts**

- Sort & flush buffer
- Sort-merge & Eliminate duplicates

**Diagram:**

- Buffer levels: 0, 1, 2, 3
- Sorted arrays
- Inserts: $X_1$, ..., $X_2$, ...
- Sort-merge & Eliminate duplicates
Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

*Design principle #2:* optimize for lookups by sort-merging arrays

Design principle #1:
- Buffer (Level 0)
- Sorted arrays (Levels 1, 2, 3)

Inserts:
- sort & flush buffer
  - $X_1 \ldots X_2 \ldots$

Sort-merge & Eliminate duplicates
**Basic LSM-tree**

*Design principle #1:* optimize for insertions by buffering

*Design principle #2:* optimize for lookups by sort-merging arrays

![Diagram of LSM-tree with levels and buffer](image)

- **Buffer**
  - Level 0
  - Level 1
  - Level 2
  - Level 3

Inserts:
- Sort & flush buffer
- Sort-merge & Eliminate duplicates & Discard original arrays
Basic LSM-tree – Example
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4 6 9
Basic LSM-tree – Example

Buffer

Sorted arrays

Level
0
1
2
3

inserts

sort & flush buffer

4 6 9
Basic LSM-tree – Example

Buffer

Sorter arrays

Level

0

1

2

3

inserts

4

6

9
Basic LSM-tree – Example

Level
Buffer

Sorted arrays

inserts

3 4 8
4 6 9
Basic LSM-tree – Example

- Buffer
  - Level 0
  - Level 1
  - Level 2
  - Level 3

- Inserts

- Sort & Flush Buffer
Basic LSM-tree – Example

```
Buffer
Level
0
1
2
3
Sorted arrays
```

```
<table>
<thead>
<tr>
<th>Level</th>
<th>inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4 6 9</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 4 8</td>
</tr>
</tbody>
</table>
```
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

Sort-merge

4 6 9

3 4 8

3 4 6 8 9
Basic LSM-tree – Example

- Buffer
  - Sorted arrays
  - Level 0
  - Level 1
  - Level 2
  - Level 3

Inserts

Sort-merge & Eliminate duplicates
Basic LSM-tree – Example

- **Buffer**
  - Level 0
  - Level 1
  - Level 2
  - Level 3

- **Sorted arrays**

- **inserts**
  - 4
  - 6
  - 9
  - 3
  - 4
  - 8
  - 9

- **Sort-merge & Eliminate duplicates & Discard original arrays**
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

3 4 6 8 9
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

2 7 8

3 4 6 8 9
Basic LSM-tree – Example

- **Buffer**
  - Level 0
  - Level 1
  - Level 2
  - Level 3

- **Sorted arrays**

- **Inserts**
  - Level 0: 3 4 5
  - Level 1: 6 7 8
  - Level 2: 9

- **Sort & flush buffer**
Basic LSM-tree – Example

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

inserts:

- Buffer: 0, 1, 2, 3
- Sorted arrays: 2, 7, 8, 3, 4, 6, 8, 9
# Basic LSM-tree

Levels have exponentially increasing capacities.

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer 0</td>
<td>1</td>
</tr>
<tr>
<td>Buffer 1</td>
<td>2</td>
</tr>
<tr>
<td>Sorted arrays 2</td>
<td>4</td>
</tr>
<tr>
<td>Sorted arrays 3</td>
<td>8</td>
</tr>
</tbody>
</table>
# Basic LSM-tree – Lookup cost

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

**Lookup method?**
- Search youngest to oldest.

**How?**
- Binary search.

**Lookup cost?**

- \( \mathcal{O} \left( \log_2 \left( \frac{N}{B} \right) \right) \)
- \( \mathcal{O} \left( \log_2 \left( \frac{N}{B} \right)^2 \right) \)

Capacity

<table>
<thead>
<tr>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
Basic LSM-tree – Insertion cost

How many times is each entry copied?

What is the price of each copy?

Total insert cost?

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>8</td>
</tr>
</tbody>
</table>

$$0 \left( \frac{\log_2 \left( \frac{N}{B} \right)}{} \right)$$

$$0 \left( \frac{1}{B} \right)$$

$$0 \left( \frac{1}{B} \cdot \log_2 \left( \frac{N}{B} \right) \right)$$
### Results Catalogue

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Results Catalogue

Better *insert cost* and *worst lookup cost* compared with B-trees

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results Catalogue

**Better insert cost** and **worst lookup cost** compared with B-trees
Can we improve lookup cost?

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Declining Main Memory Cost
Declining Main Memory Cost

Store a fence pointer for every block in main memory
Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sorted array</strong></td>
<td>$O(\log_2(N/B))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td><strong>Leveled LSM-tree</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tiered LSM-tree</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N/B))$</td>
<td>$O(\log_B(N/B))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>O(1)</td>
<td>O(N/B)</td>
</tr>
<tr>
<td>Log</td>
<td>O(N/B)</td>
<td>O(1/B)</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>O(log₂(N/B)^2)</td>
<td>O(1/B · log₂(N/B))</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th>Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B)^2)$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B))$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results Catalogue – with fence pointers

Quick sanity check: suppose and $N = 2^{42}$
and $B = 2^{10}$

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N/B))$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results Catalogue – with fence pointers

### Quick sanity check:
Suppose $N = 2^{42}$ and $B = 2^{10}$

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(2^{32})$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(2^{32})$</td>
<td>$O(2^{-10})$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(5)$</td>
<td>$O(2^{-10} \cdot 5)$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Leveled LSM-tree

- Lookup cost
- Update cost
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?

Increase size ratio $T$

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>$T^0$</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>$T^1$</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>$T^2$</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>$T^3$</td>
</tr>
</tbody>
</table>
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio $T$

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
</tbody>
</table>
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio $T$

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
</tbody>
</table>

Buffer

Sorted arrays
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio $T$

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>64</td>
</tr>
</tbody>
</table>
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it? Increase size ratio $T$
E.g. size ratio of 4

![Diagram of Level 0, Level 1, Level 2, Level 3 with flush & sort-merge and capacity values: 1, 4, 16, 64]
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T

Buffer

Sorted arrays

Level

Capacity

0
1
2
3

... ... ...

... ... ...

... ... ...

... ... ...

... ... ...

... ... ...

1
4
16
64
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it? E.g. size ratio of 4

Increase size ratio T

Buffer

Sorted arrays

Level
0
1
2
3

Capacity
1
4
16
64

Capacity
1
4
16
64

flush & sort-merge

inserts
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio $T$

- Buffer
- Sorted arrays

Level 0
- Capacity 1

Level 1
- Capacity 4

Level 2
- Capacity 16

Level 3
- Capacity 64

Inserts

Move
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Capacity

Buffer

Sorted arrays

Level

0

1

2

3

Capacity

1

4

16

64

Inserts

Capacity
Leveled LSM-tree

Lookup cost?
$O \left( \log_T \left( \frac{N}{B} \right) \right)$

Insertion cost?
$O \left( \frac{T}{B} \cdot \log_T \left( \frac{N}{B} \right) \right)$

Level 0
- Buffer
- Sorted arrays

Level 1

Level 2

Level 3

Capacity
1
4
16
64
Leveled LSM-tree

Lookup cost?
\[ O \left( \log_T \left( \frac{N}{B} \right) \right) \]

Insertion cost?
\[ O \left( \frac{T}{B} \cdot \log_T \left( \frac{N}{B} \right) \right) \]

What happens as we increase the size ratio \( T \)?

What happens when size ratio \( T \) is set to be \( N/B \)?

Lookup cost becomes:
\[ O(1) \]

Insert cost becomes:
\[ O\left(\frac{N}{B^2}\right) \]

The LSM-tree becomes a sorted array!
lookup cost vs. insertion cost

- Basic
- LSM-tree
- Sorted array

Leveling
## Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>$O(\log_T(N/B))$</td>
<td>$O(T/B \cdot \log_T(N/B))$</td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tiered LSM-tree

↑ Lookup cost

↓ Insertion cost
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
</tbody>
</table>

Buffer

Sorted arrays
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio. Do not merge within a level. E.g. size ratio of 4
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4

Buffer

Sorted arrays

Level

0
1
2
3

Capacity

1
4
16
64

Capacity

1
4
16
64
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio. Do not merge within a level. E.g. size ratio of 4

Buffer

Sorted arrays

Level

0

1

2

3

Capacity

1

4

16

64
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4

<table>
<thead>
<tr>
<th>Level</th>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inserts

Flush

Capacity

1
4
16
64
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4
Tiered LSM-tree

Lookup cost?
\[ 0 \left( T \cdot \log_T \left( \frac{N}{B} \right) \right) \]

Insertion cost?
\[ 0 \left( \frac{1}{B} \cdot \log_T \left( \frac{N}{B} \right) \right) \]

Level  
Buffer  
0  
1  
2  
3  
Sorted arrays

Capacity
1
4
16
64
Tiered LSM-tree

Lookup cost?
$O\left(T \cdot \log_T \left(\frac{N}{B}\right)\right)$

Insertion cost?
$O\left(\frac{1}{B} \cdot \log_T \left(\frac{N}{B}\right)\right)$

What happens as we increase the size ratio $T$?

What happens when size ratio $T$ is set to be $N/B$?

Lookup cost becomes:
$O(N/B)$

Insert cost becomes:
$O(1/B)$

The tiered LSM-tree becomes a log!
Results Catalogue – with fence pointers

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N/B)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N/B))$</td>
<td>$O(1/B \cdot \log_2(N/B))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>$O(\log_T(N/B))$</td>
<td>$O(T/B \cdot \log_T(N/B))$</td>
</tr>
<tr>
<td><strong>Tiered LSM-tree</strong></td>
<td>$O(T \cdot \log_T(N/B))$</td>
<td>$O(1/B \cdot \log_T(N/B))$</td>
</tr>
</tbody>
</table>
Bloom filters
Declining Main Memory Cost
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for X

filters

\[
\begin{array}{cccccccc}
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & X & \ldots & \ldots & \ldots
\end{array}
\]

array
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for X

![Filters](filters)

![Array](array)

... ... ... ... ... ... ... X ... ... ...
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for Y

filters

array

... ... ... ... ... ... ... X ... ... ...
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for Y

filters

array
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for $Y$
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for Z

... ... ... ... ... ... X ... ...

filters

array
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for Z

filters

array

... ... ... ... ... ... ... X ... ... ...
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.
Bloom Filters

The more main memory, the fewer false positives $\implies$ cheaper lookups
Bloom Filters

The more main memory, the fewer false positives $\Rightarrow$ cheaper lookups
Conclusions

Write-optimized

Highly tunable

Backbone of many modern systems

Trade-off between lookup and insert cost (tiering/leveling, size ratio)

Trade main memory for lookup cost (fence pointers, Bloom filters)

Thank you!
CS460: Intro to Database Systems

Database Systems and Beyond

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/
Database Systems

we spent a whole semester on Database Systems
what is next?

what can we do with data?

data-driven science          data-driven discovery

data-driven governance
“Experimental, theoretical, and computational science are all being affected by the data deluge, and a fourth, ‘data-intensive’ science paradigm is emerging.

The goal is to have a world in which all of the science literature is online, all of the science data is online, and they interoperate with each other. **Lots of new tools are needed to make this happen.**

Faster Innovation through Data-Intensive Approaches

Need for Innovation in Data Management!
The Backbone is Database Systems, Storage Engines, & Frameworks for Parallelization

increase **throughput** by parallelization

“scale-up”
use more powerful machines (>\#CPUs, >RAM)

“scale-out”
use more machines
Scale Up Execution

how to use more cores (threads)?

**inter-query parallelism**

each query runs on one processor

**inter-operator parallelism**

each query runs on multiple processors
an operator runs on one processor

**intra-operator parallelism**

An operator runs on multiple processors
Scale Up Storage

needs more disks!

how to distribute data?

block partition
hash partition
range partition

how to distribute data accesses?
Scale Out

similar questions across machines

new bottlenecks?

move data across machines: network!
Versatile and popular infrastructure: NoSQL stores
dancing into the internals of modern data systems

cutting-edge designs / research projects / engineering projects

CS 561: Data Systems Architectures

Spring 2021
A path in data science

(1) strong data systems skills
   (i) coding skills
   (ii) system architecture insights
        performance tradeoffs

(2) application domain knowledge

(3) statistics, machine learning, math tools
Academic Research
Industry