CS460: Intro to Database Systems

Class 16: 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
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

- **Main Memory**
  - Metadata & frequently accessed data
  - Expensive, fast

- **Disk**
  - All data
  - Cheap, slow
Why is disk slow?

Random access is slow  $\Rightarrow$  move disk head
Sequential access is faster  $\Rightarrow$  let disk spin
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 ➔ 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>
Sorted Array

\( n \) entries

\( B \) entries fit into a disk block

Array spans \( N = \frac{n}{B} \) disk blocks

Measure Performance in I/Os

Lookup method & cost?

Binary search: \( O(\log_2(N)) \) I/Os

Insertion cost?

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

Buffer

| James | Sara |

Array size | Pointer

Block 1 | Block 2 | ... | Block N

| 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>$O(\log_2(N))$</td>
<td>$O(N/B)$</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>$O(\log_2(N))$</td>
<td>$O(N/B)$</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)

\( n \) entries

\( B \) entries fit into a disk block

Array spans \( N = \frac{n}{B} \) disk blocks

Lookup method & cost?

- **Scan:** \( O(N) \)
- **Insertion cost:**
  - **Append:** \( O\left(\frac{1}{B}\right)\)

---

**Buffer**

- James
- Sara

---

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N</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>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$O(N)$</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>Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N)$</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: $O(\log_B(N))$

Insertion method & cost?
Tree search & append: $O(\log_B(N))$

Depth: $O(\log_B(N))$
## 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>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</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 storage
Workloads more insert-intensive
We need better insert-performance
## Results Catalogue

### Goal to combine

- sub-constant insertion cost
- logarithmic lookup cost

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</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>
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

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

Sorted arrays

Inserts

Basic LSM-tree

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

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

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

-----

Design principle #1: optimize for insertions by buffering

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

Inserts

Level

Buffer

Sorted arrays

sort & flush buffer

Sort-merge
Basic LSM-tree

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

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

![Diagram of Basic LSM-tree]

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

- **Sorted arrays**

**Inserts**

- sort & flush buffer

- 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

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

- Sorted arrays

**Inserts**

- sort & flush buffer

- 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

---

**Buffer**

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

---

**Sort-merge & Eliminate duplicates & Discard original arrays**

**Inserts**

- sort & flush buffer

---

[Diagram showing levels of buffer and sorted arrays, with arrows indicating insertion and sorting processes.]
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3
Basic LSM-tree – Example

Level

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4 9
Basic LSM-tree – Example

- Buffer
- Sorted arrays

Level

0
1
2
3

Inserts:

4 9 6

Sort
Basic LSM-tree – Example

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

- inserts

- sort

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
{ 0
  1
  2
  3
}

Sorted arrays

inserts
↓

4 6 9
Basic LSM-tree – Example

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Sorted arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>0</td>
<td>3 8 4</td>
</tr>
<tr>
<td>1</td>
<td>4 6 9</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

inserts
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0
1
2
3

inserts

sort & flush buffer

4 6 9
3 4 8
Basic LSM-tree – Example

Buffer
{ 0
  1
  2
  3
}

Sorted arrays

Level

inserts

4 6 9
3 4 8
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4 6 9

3 4 8

3 4 6 8 9

Sort-merge
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

3

4

6

9

3

4

8

3

4

6

8

9

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

Buffer

Sorted arrays

Level

0

1

2

3

inserts

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

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

- Sorted arrays

- 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

Sorted arrays

Level

0

1

2

3

inserts

sort & flush buffer

2 7 8

3 4 6 8 9
Basic LSM-tree – Example

Buffer

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>2 7 8</td>
</tr>
<tr>
<td>2</td>
<td>3 4 6 8 9</td>
</tr>
</tbody>
</table>
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>Buffer 2</td>
<td>4</td>
</tr>
<tr>
<td>Sorted arrays 3</td>
<td>8</td>
</tr>
</tbody>
</table>

Levels have exponentially increasing capacities.
Basic LSM-tree – Lookup cost

**Lookup method?**
Search youngest to oldest.
Binary search.

**How?**

**Lookup cost?**

<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>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ O(\log_2(N)) \]
\[ O(\log_2(N)) \]
\[ O(\log_2(N)^2) \]
Basic LSM-tree – Insertion cost

How many times is each entry copied?

What is the price of each copy?

Total insert cost?

0(\log_2(N))

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

0\left(\frac{1}{B} \cdot \log_2(N)\right)

How many times is each entry copied?

What is the price of each copy?

Total insert cost?
## Results Catalogue

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

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

![Graph showing the decline in main memory cost over time. The x-axis represents the year, ranging from 1980 to 2015. The y-axis represents the price per GB ($), with a logarithmic scale from $10^{-5}$ to $10^9$. The graph includes data points for both main memory and disk over the years, showing a consistent decline in cost.](image-url)
Declining Main Memory Cost

Store a fence pointer for every block in main memory
## 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>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</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>$O(\log_2(N))$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</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></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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lookup cost</td>
<td>Insertion cost</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Sorted array</td>
<td>$O(1)$</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>Log</td>
<td>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N)^2)$</td>
<td>$O(1/B \cdot \log_2(N))$</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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>$O(\log_2(N))$</td>
<td>$O(1/B \cdot \log_2(N))$</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 \( N = 2^{32} \) 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(N/B) )</td>
</tr>
<tr>
<td>Log</td>
<td>( O(N) )</td>
<td>( O(1/B) )</td>
</tr>
<tr>
<td>B-tree</td>
<td>( O(\log_B(N)) )</td>
<td>( O(\log_B(N)) )</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>( O(\log_2(N)) )</td>
<td>( O(1/B \cdot \log_2(N)) )</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 \( N = 2^{32} \) 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(2^{22}) )</td>
</tr>
<tr>
<td>Log</td>
<td>( O(2^{32}) )</td>
<td>( O(2^{-10}) )</td>
</tr>
<tr>
<td>B-tree</td>
<td>( O(4) )</td>
<td>( O(4) )</td>
</tr>
<tr>
<td><strong>Basic LSM-tree</strong></td>
<td>( O(32) )</td>
<td>( O(2^{-10} \cdot 32) )</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 $T$

Capacity

$T^0$

$T^1$

$T^2$

$T^3$
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

<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

Buffer

Sorted arrays

Level

0
1
2
3

Capacity

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
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

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 \( T \)

Inserts

Flush & sort-merge

Level

Capacity

Buffer

Sorted arrays

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

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 $T$

Inserts

Flush & sort-merge

Buffer

Sorted arrays

Level

0
1
2
3

Capacity

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

lookup cost depends on number of levels
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?  Increase size ratio $T$
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>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

Capacity

Capacity

Capacity
Leveled LSM-tree

Lookup cost?
$O(\log_T(N))$

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

Buffer
- Level 0:...
- Level 1:...
- Level 2:...
- Level 3:...

Capacity
1 4 16 64

Sorted arrays
Leveled LSM-tree

Lookup cost?
\[ O( \log_T (N) ) \]

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

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

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

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

Insert cost becomes:
\[ O(N/B) \]

The LSM-tree becomes a sorted array!
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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N))$</td>
<td>$O(1/B \cdot \log_2(N))$</td>
</tr>
<tr>
<td><strong>Leveled LSM-tree</strong></td>
<td>$O(\log_T(N))$</td>
<td>$O(T/B \cdot \log_T(N))$</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

- Buffer
- Sorted arrays

<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>
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>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Capacity

- Level 0: 1
- Level 1: 4
- Level 2: 16
- Level 3: 64

inserts

flush
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

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

<table>
<thead>
<tr>
<th>Level</th>
<th>Insert</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer 0</td>
<td>... ... ...</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>... ... ... ... ... ... ...</td>
<td>4</td>
</tr>
<tr>
<td>Sorted arrays 2</td>
<td>... ... ... ... ... ... ...</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>... ... ... ... ... ... ... ... ... ...</td>
<td>64</td>
</tr>
</tbody>
</table>
## 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>Inserts</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>64</td>
</tr>
</tbody>
</table>

Reduce the number of levels by increasing the size ratio.
Tiered LSM-tree

**Lookup cost?**

\[ O(T \cdot \log_T(N)) \]

**Insertion cost?**

\[ O\left(\frac{1}{B} \cdot \log_T(N)\right) \]

<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:

- Capacity:
  - 1
  - 4
  - 16
  - 64
Tiered LSM-tree

Lookup cost?
$O(T \cdot \log_T(N))$

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

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

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

Lookup cost becomes:
$O(N)$

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

The tiered LSM-tree becomes a log!
Lookup cost

Insertion cost

Log

Tiering

Basic LSM-tree

Leveling

Sorted array
## 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>$O(N)$</td>
<td>$O(1/B)$</td>
</tr>
<tr>
<td>B-tree</td>
<td>$O(\log_B(N))$</td>
<td>$O(\log_B(N))$</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>$O(\log_2(N))$</td>
<td>$O(1/B \cdot \log_2(N))$</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>$O(\log_T(N))$</td>
<td>$O(T/B \cdot \log_T(N))$</td>
</tr>
<tr>
<td><strong>Tiered LSM-tree</strong></td>
<td>$O(T \cdot \log_T(N))$</td>
<td>$O(1/B \cdot \log_T(N))$</td>
</tr>
</tbody>
</table>
Results Catalogue – with fence pointers

Quick sanity check: suppose and and

<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)</td>
<td>O(1/B)</td>
</tr>
<tr>
<td>B-tree</td>
<td>O(logB(N))</td>
<td>O(logB(N))</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>O(log2(N))</td>
<td>O(1/B \cdot log2(N))</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>O(logT(N))</td>
<td>O(T/B \cdot logT(N))</td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td>O(T \cdot logT(N))</td>
<td>O(1/B \cdot logT(N))</td>
</tr>
</tbody>
</table>

Results Catalogue – with fence pointers

Quick sanity check: suppose and and

<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)</td>
<td>O(1/B)</td>
</tr>
<tr>
<td>B-tree</td>
<td>O(logB(N))</td>
<td>O(logB(N))</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>O(log2(N))</td>
<td>O(1/B \cdot log2(N))</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>O(logT(N))</td>
<td>O(T/B \cdot logT(N))</td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td>O(T \cdot logT(N))</td>
<td>O(1/B \cdot logT(N))</td>
</tr>
</tbody>
</table>

```ruby
N = 2^{32}
B = 2^{10}
T = 2^2
```
Results Catalogue – with fence pointers

Quick sanity check:

Suppose

\[ N = 2^{32} \]
\[ B = 2^{10} \]
\[ T = 2^2 \]

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Insertion cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>(2^0=1)</td>
<td>(2^{22}=4K)</td>
</tr>
<tr>
<td>Log</td>
<td>(2^{32}=4M)</td>
<td>(2^{-10}=0.001)</td>
</tr>
<tr>
<td>B-tree</td>
<td>(2^2=4)</td>
<td>(2^2=4)</td>
</tr>
<tr>
<td>Basic LSM-tree</td>
<td>(2^5=32)</td>
<td>(2^{-5}=0.031)</td>
</tr>
<tr>
<td>Leveled LSM-tree</td>
<td>(2^4=16)</td>
<td>(2^{-4}=0.063)</td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td>(2^6=64)</td>
<td>(2^{-6}=0.016)</td>
</tr>
</tbody>
</table>
Bloom filters
Declining Main Memory Cost

- **Main Memory**
- **Disk**
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

Bloom filter

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 X

filters

Bloom filter

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.

```plaintext
Lookup for X

Access on disk
```

```
filters

array
```

```plaintext
... ... ... ... ... ... ... 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

... ... ... ... ... ... ... 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

Bloom filter

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

filters

array

Bloom filter

... ... ... ... ... ... ... 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 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.

Lookup for Z

Bloom filter

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!