CS460: Intro to Database Systems

Class 14: Log-Structured-Merge Trees

Instructor: Manos Athanassoulis

https://midas.bu.edu/classes/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

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

- **Disk**
  - cheap, slow
  - All data
≈ 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

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

Approximately 100 ns

4 kilobyte Blocks

Approximately 10 ms

64 byte Words

Approximately 1 ns
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></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>$O(N/B^2)$</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)

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?

**Scan:** \( O\left(\frac{N}{B}\right) \)

Insertion cost?

**Append:** \( O\left(\frac{1}{B}\right) \)
## 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><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></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>Dataset Type</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>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></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 sub-constant insertion cost logarithmic lookup cost

<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>$O(N/B^2)$</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>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
Basic LSM-tree

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

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

- **Level 0:** Buffer
- **Level 1:** Sorted arrays
- **Level 2:** Inserts
- **Level 3:** ...
- ...
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

Inserts

sort & flush buffer

Buffer

Sorted arrays

Level

0

1

2

3
Basic LSM-tree

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

![Diagram of Basic LSM-tree]

- 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

---

**Buffer**

- **Level 0**
  - Sorted arrays

- **Level 1**
  - ...

- **Level 2**
  - ...

- **Level 3**
  - ...

**Inserts**

- 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

 Inserts 

 Buffer  

 Sorted arrays  

 Level  

 0  1  2  3 

 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
Basic LSM-tree

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

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

![Diagram]

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

- **Sorted arrays:**

- **Inserts**
  - Sort & flush buffer

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

Buffer

Sorted arrays

Level

0

1

2

3
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

Level

Buffer

0

1

Sorted arrays

2

3

inserts

4 6 9
Basic LSM-tree – Example

Buffer

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

Sorted arrays

inserts

3 4 8
4 6 9
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
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

Inserts

4 6 9

3 4 8

Sort-merge
Basic LSM-tree – Example

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

Sorted arrays

Inserts:
- $4_1 \ 6 \ 9$
- $3 \ 4_2 \ 8$
- $3 \ 4_2 \ 6 \ 8 \ 9$

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

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4 6 9

3 4 6 8 9

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

Level

Buffer

Sorted arrays

Level

Level

Level

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

Level

0

1

2

3

Sorted arrays

inserts

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</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
| Sorted arrays | 0 | ...
|         | 1 | ...
|         | 2 | ...
|         | 3 | ...
Basic LSM-tree – Lookup cost

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

**How?**
Binary search.

**Lookup cost?**

<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>

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

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

0 \left( \log_2 \left( \frac{N}{B} \right)^2 \right)

Capacity

1
2
4
8
Basic LSM-tree – Insertion cost

How many times is each entry copied?

What is the price of each copy?

Total insert cost?

\[
\begin{align*}
\text{Level} & \quad \text{Capacity} \\
\text{Buffer} & \quad 1 \\
\text{Sorted arrays} & \quad 2 \\
\quad 0 & \quad 1 \\
\quad 1 & \quad 2 \\
\quad 2 & \quad 4 \\
\quad 3 & \quad 8 \\
\end{align*}
\]

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

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

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

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><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

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>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>$O(N/B)$</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>
Declining Main Memory Cost

![Graph showing the declining cost of main memory and disk over time. The x-axis represents the year from 1980 to 2015, and the y-axis represents the price per GB in dollars on a logarithmic scale. The graph shows a significant decrease in price over the years.]
Declining Main Memory Cost

Store a fence pointer for every block in main memory

Fence pointers

array

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>18</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>$O(\log_2(N/B))$</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></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>
<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><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>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></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>$O(\log_2(N/B)^2)$</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>Dataset</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></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>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>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>$T^0$</td>
</tr>
<tr>
<td>$T^1$</td>
</tr>
<tr>
<td>$T^2$</td>
</tr>
<tr>
<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>Buffer</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Sorted arrays</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>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>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?
E.g. size ratio of 4

Increase size ratio T

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

Capacity

0

1

2

3

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 $T$

Flush & sort-merge

<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 $T$

inserts

flush & sort-merge

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

inserts

move
Leveled LSM-tree

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

Increase size ratio $T$

Level

Buffer

Sorted arrays

Level 0
Level 1
Level 2
Level 3

Capacity

1
4
16
64

Capacity

1
4
16
64

Inserts
Leveled LSM-tree

Lookup cost?
$$0 \left( \log_T \left( \frac{N}{B} \right) \right)$$

Insertion cost?
$$0 \left( \frac{T}{B} \cdot \log_T \left( \frac{N}{B} \right) \right)$$

<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>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>

inserts
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:** $O(N/B^2)$

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>$O(N/B)$</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><strong>Leveled LSM-tree</strong></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>
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>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

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

Buffer

Level

0
1
2
3

Sorted arrays

Capacity

1
4
16
64

inserts

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

sort-merge
## 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>Buffer</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Sorted</td>
<td>2</td>
</tr>
<tr>
<td>arrays</td>
<td>3</td>
</tr>
</tbody>
</table>

```
... ... ...
... ... ...
... ... ...
... ... ...
```
Tiered LSM-tree

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

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

Level

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>

inserts
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>Tiered LSM-tree</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

![Graph showing the declining cost of main memory over years.](https://midas.bu.edu/classes/CS460/ManosAthanasoulis)
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

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 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.
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.
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 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

filters

array

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

Bloom filter
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
Access on disk
Bloom Filters

The more main memory, the fewer false positives \(\Rightarrow\) 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!