Log-Structured-Merge Trees

Comp115 guest lecture
Niv Dayan
23 February, 2017
Useful when?

- Massive dataset
- Rapid updates/insertions
- Fast lookups

⇒ LSM-trees are for you.
Invented in 1996
Patrick O'Neil
UMass Boston

Invented in 1996

Time
1980 1990 2000 2010
Patrick O'Neil
UMass Boston

Invented in 1996
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
Main Memory

expensive, fast

Disk

cheap, slow
The Memory Hierarchy

- **CPU**
  - Main Memory: expensive, fast
  - Disk: cheap, slow
The Memory Hierarchy

- **Main Memory**: expensive, fast
- **Disk**: cheap, slow

- **Metadata & frequently accessed data**
- **All data**
≈ 100 ns

≈ 10 ms

≈ 5-6 order of magnitude difference
≈ 100 ns

≈ 10 ms

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

Disk head
Why is disk slow?

Random access is slow  ➞  move disk head
Sequential access is faster  ➞  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
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: \(\approx 1 \text{ ns}\)
- 4 kilobyte Blocks: \(\approx 100 \text{ ns}\)
- Hard disk: \(\approx 10 \text{ ms}\)
Modeling Performance

64 byte Words

4 kilobyte Blocks

Measure bottleneck:

Number of block reads/writes (I/O)

Approximate:

\[
\text{1 ns}
\]

\[
\text{100 ns}
\]

\[
\text{10 ms}
\]
Sorted Array

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anne</td>
<td>Bob</td>
<td></td>
<td>Yulia</td>
</tr>
<tr>
<td>Arnold</td>
<td>Corrie</td>
<td></td>
<td>Zack</td>
</tr>
<tr>
<td>Barbara</td>
<td>Doug</td>
<td></td>
<td>Zelda</td>
</tr>
</tbody>
</table>

Buffer
- James
- Sara
Sorted Array

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

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

Lookup method & cost?

---

**Buffer**
- James
- Sara

**Array size** | **Pointer**
--- | ---
Anne | Yulia
Arnold | Zack
Barbara | Zelda
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

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Array size</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sara</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anne</td>
<td>Bob</td>
<td></td>
<td>Yulia</td>
</tr>
<tr>
<td>Arnold</td>
<td>Corrie</td>
<td></td>
<td>Zack</td>
</tr>
<tr>
<td>Barbara</td>
<td>Doug</td>
<td></td>
<td>Zelda</td>
</tr>
</tbody>
</table>
Sorted Array

N entries
B entries fit into a disk block
Array spans $\frac{N}{B}$ disk blocks

Lookup method & cost?
Binary search: $O \left( \log_2 \left( \frac{N}{B} \right) \right)$ I/Os
Insertion cost?

Buffer
- James
- Sara

Array size
- Block 1: Anne
- Block 2: Arnold
- ...: Barbara
- Block N/B: Yulia

Pointer
- Block 1: Bob
- Block 2: Corrie
- ...: Doug
- Block N/B: Zack

Download
Download
Download
Sorted Array

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

Lookup method & cost?
Binary search: $0 \left( \log_2 \left( \frac{N}{B} \right) \right)$ I/Os
Insertion cost?
Push entries: $0 \left( \frac{1}{B} \cdot \frac{N}{B} \right)$ 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)

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Array size</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sara</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/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>
Log  (append-only array)

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Array size</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sara</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/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>
Log (append-only array)

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

Lookup method & cost?

Buffer
James
Sara

Array size  Pointer

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/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>
Log (append-only array)

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

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

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Array size</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Block 1</td>
<td>Doug</td>
</tr>
<tr>
<td>Sara</td>
<td>Block 2</td>
<td>Yulia</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Block N/B</td>
<td>Anne</td>
</tr>
<tr>
<td>Zelda</td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>Zack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arnold</td>
<td></td>
<td>Corrie</td>
</tr>
<tr>
<td>Barbara</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?

Buffer
- James
- Sara

Array size

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/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>
Log (append-only array)

\( N \) entries

\( B \) entries fit into a disk block

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

Lookup method & cost?

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

Buffer

<table>
<thead>
<tr>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
</tr>
<tr>
<td>Sara</td>
</tr>
</tbody>
</table>

Array size

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>...</th>
<th>Block N/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></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
B-tree

Lookup method & cost?
B-tree

Lookup method & cost?

Depth: $O(\log_B(N/B))$
B-tree

Lookup method & cost?

Tree search: \(0 \left( \log_B \left( \frac{N}{B} \right) \right)\)

Depth: \(O(\log_B(N/B))\)
B-tree

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

Insertion method & cost?

Depth: $O(\log_B(N/B))$
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 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>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>
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
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

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

Inserts

Level

Buffer

Sorted arrays

0
1
2
3

sort & flush buffer
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](image)
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
- Level 1
- Level 2
- Level 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

---

Buffer

Sorted arrays

Level

0
1
2
3

Inserts

\[X_1 \ldots \ldots \]

sort & flush buffer

\[\ldots X_2 \ldots \]

Sort-merge & Eliminate duplicates

\[\ldots X_2 \ldots \ldots \ldots \]
Basic LSM-tree

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

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

Level

0
1
2
3

inserts

↓

4 6 9
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

3

4

8

4

6

9

inserts
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

sort & flush buffer
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4 6 9

3 4 8
Basic LSM-tree – Example

Level

Buffer

Sorted
arrays

inserts

Sort-merge
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

4_1 6 9

3 4_2 8

3 4_2 6 8 9

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

The diagram illustrates a basic LSM-tree example. It consists of multiple levels, each containing sorted arrays. The process involves adding values (inserts) to the buffer, sorting them, and then merging them into the sorted arrays. Duplicates are eliminated, and the original arrays are discarded.

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

Inserts: 3 4 6 9

Sort-merge and eliminate duplicates, then 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

inserts

0

1

2

3

2 7 8

3 4 6 8 9
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

sort & flush buffer

\[ \begin{array}{c}
\text{Buffer:} \\
0 \\
1 \\
2 \\
3 \\
\end{array} \]

\[ \begin{array}{cccccccc}
\text{Sorted arrays:} \\
& \text{2} & \text{7} & \text{8} \\
& \text{3} & \text{4} & \text{6} & \text{8} & \text{9} \\
\end{array} \]
Basic LSM-tree – Example

Buffer

Sorted arrays

Level

0

1

2

3

inserts

2 7 8

3 4 6 8 9
Basic LSM-tree

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

Buffer
0
1
2
3

Sorted arrays

Level

Capacity
1
2
4
8

Lookup method?
Search youngest to oldest.

Lookup cost?
$O(\log)$
Basic LSM-tree – Lookup cost

**Lookup method?**

<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>

**Capacity**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

O(log N) & (N)”

*How?*

Binary search.
Basic LSM-tree – Lookup cost

**Lookup method?**

Search youngest to oldest. $O \left( \log_2 \left( \frac{N}{B} \right) \right)$
Basic LSM-tree – Lookup cost

**Lookup method?**

Search youngest to oldest. \( O \left( \log_2 \left( \frac{N}{B} \right) \right) \)

**How?**

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

- **Sorted arrays**
  - Level 0
    - ... ... ...
  - Level 1
    - ... ... ...
  - Level 2
    - ... ... ... ...
  - Level 3
    - ... ... ... ... ... ... ...

**Capacity**

- 1
- 2
- 4
- 8
Basic LSM-tree – Lookup cost

**Lookup method?**

Search youngest to oldest. $O\left(\log_2 \left(\frac{N}{B}\right)\right)$

**How?**

Binary search. $O\left(\log_2 \left(\frac{N}{B}\right)\right)$

---

**Level**

- **Buffer**
  - Level 0:
    - Capacity 1
  - Level 1:
    - Capacity 2
  - Level 2:
    - Capacity 4
  - Level 3:
    - Capacity 8
Basic LSM-tree – Lookup cost

**Lookup method?**
Search youngest to oldest. \( O \left( \log_2 \left( \frac{N}{B} \right) \right) \)

**How?**
Binary search. \( O \left( \log_2 \left( \frac{N}{B} \right) \right) \)

**Lookup cost?**

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer 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>

- Buffer
- Sorted arrays
## 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>

**Capacity**
- $O\left(\log_2 \left(\frac{N}{B}\right)\right)$
- $O\left(\log_2 \left(\frac{N}{B}\right)^2\right)$
Basic LSM-tree – Insertion cost

### Buffer
- Level 0: 
  - Capacity: 1
  - Entries: ...
- Level 1: 
  - Capacity: 2
  - Entries: ...
- Level 2: 
  - Capacity: 4
  - Entries: ...
- Level 3: 
  - Capacity: 8
  - Entries: ...

### Sorted arrays
- Capacity: 1
- Entries: ...
- Capacity: 2
- Entries: ...
- Capacity: 4
- Entries: ...
- Capacity: 8
- Entries: ...
Basic LSM-tree – Insertion cost

*How many times is each entry copied?*

<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>
Basic LSM-tree – Insertion cost

How many times is each entry copied? \[ O \left( \log_2 \left( \frac{N}{B} \right) \right) \]

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

*How many times is each entry copied?*  
$O\left(\log_2 \left(\frac{N}{B}\right)\right)$

*What is the price of each copy?*

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Sorted arrays 2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<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?

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

0 \left( \frac{1}{B} \right)
Basic LSM-tree – Insertion cost

How many times is each entry copied? \( O \left( \log_2 \left( \frac{N}{B} \right) \right) \)

What is the price of each copy? \( O \left( \frac{1}{B} \right) \)

Total insert 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>

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

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

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

\[
O \left( \frac{1}{B} \cdot \log_2 \left( \frac{N}{B} \right) \right)
\]
## 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>
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

Price per GB ($)

Year


10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^0 10^1 10^2 10^3 10^4 10^5 10^6 10^7 10^8 10^9

Main Memory
Disk
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>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>
## 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>$O(\log_2(N/B))$</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>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 · 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><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>

The table above compares the lookup and insertion costs for different data structures, including sorted arrays, logs, B-trees, and different types of LSM-trees. The costs are given in Big O notation, indicating the upper bound on the time complexity as the data size grows.
## 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>$O(\log_B(N/B))$</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><strong>$O(1)$</strong></td>
<td><strong>$O(1)$</strong></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>$O(1)$</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

\[ N = 2^{42} \]
\[ B = 2^{10} \]

<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(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

\[ N = 2^{42} \]
\[ B = 2^{10} \]

<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

Buffer

Sorted arrays

Level

0

1

2

3
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
Leveled LSM-tree

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

Increase size ratio T

Level

Buffer

Level 0

... ... ...

Level 1

Level 2

Level 3

Sorted arrays
Leveled LSM-tree

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

Increase size ratio $T$

Buffer

$\begin{align*}
\text{Level} & \quad \text{Capacity} \\
0 & \quad T^0 \\
1 & \quad T^1 \\
2 & \quad T^2 \\
3 & \quad T^3
\end{align*}$

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 $T$

E.g. size ratio of 4
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 $T$

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

Sorted arrays

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 $T$

Inserts

Flush & sort-merge

Level

Buffer

Sorted arrays

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 $T$

E.g. size ratio of 4

flush & sort-merge

inserts

<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

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

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 \( T \)

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
Leveled LSM-tree

Buffer

Level

0
1
2
3

Sorted arrays

Capacity

1
4
16
64

inserts
Leveled 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></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>

capacity

- 1
- 4
- 16
- 64
Leveled LSM-tree

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

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

Insertion cost?

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

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

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

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

- Sorted arrays

Capacity
- 1
- 4
- 16
- 64
Leveled LSM-tree

Lookup cost?

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

Insertion cost?

\[ O\left( \frac{T}{B} \cdot \log_T \left( \frac{N}{B} \right) \right) \]
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 $T$?
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?
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 \( N/B \)?
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:

$$O(N/B^2)$$
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!
Basic LSM-tree
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(log₂(N/B))</td>
<td>O(1/B \cdot log₂(N/B))</td>
</tr>
<tr>
<td><strong>Leveled LSM-tree</strong></td>
<td>O(logₜ(N/B))</td>
<td>O(T/B \cdot logₜ(N/B))</td>
</tr>
<tr>
<td>Tiered LSM-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tiered LSM-tree
Tiered LSM-tree

↑ Lookup cost

↓ Insertion cost
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
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

<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>

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

- Sorted arrays

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

Sorted arrays

Capacity
- 1
- 4
- 16
- 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

<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

inserts
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

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

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

Capacity

1
4
16
64
Tiered LSM-tree

Lookup cost?

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

Sorted arrays

inserts

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

Lookup cost?

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

- Buffer
  - Level 0
  - Level 1
  - Level 2
  - Level 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?

<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>
Tiered LSM-tree

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

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

<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>

**Capacity**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tiered LSM-tree

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

Insertion cost?
\[ O \left( \frac{1}{B} \cdot \log_T \left( \frac{N}{B} \right) \right) \]
## Tiered LSM-tree

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup cost?</td>
<td>$O(T \cdot \log_T \left( \frac{N}{B} \right))$</td>
</tr>
<tr>
<td>Insertion cost?</td>
<td>$O \left( \frac{1}{B} \cdot \log_T \left( \frac{N}{B} \right) \right)$</td>
</tr>
</tbody>
</table>

What happens as we increase the size ratio $T$?
Tiered LSM-tree

**Lookup cost?**

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

**Insertion cost?**

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

What happens as we increase the size ratio \( T \)?
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?
Tiered LSM-tree

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:  $O(1/B)$
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 decline in price per GB of main memory and disk over the years from 1980 to 2015.](image)

- 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

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

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.
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

```
filter1  filter2  filter3  filter4  filter5  filter6  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 Z

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
Bloom filter
Access on disk

filters
array

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

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

The more main memory, the less false positives  ⇒ cheaper lookups
Bloom Filters

The more main memory, the less false positives $\implies$ cheaper lookups

![Graph showing the relationship between lookup cost and insertion cost.](image)
Bloom Filters

The more main memory, the less false positives \(\Rightarrow\) cheaper lookups

**Monkey:** Optimal **Navigable Key**-Value Store
Niv Dayan, Manos Athanassoulis, Stratos Idreos
SIGMOD 2017
Conclusions

Write-optimized
Conclusions

Write-optimized

Highly tunable
Conclusions

Write-optimized

Highly tunable

Backbone of many modern systems
Conclusions

- Write-optimized
- Highly tunable
- Backbone of many modern systems
- Trade-off between lookup and insert cost (tiering/leveling, size ratio)
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)
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!