Data-intensive Computing Systems

Operators for Data Access (contd.)

Shivnath Babu
Insertion in a B-Tree

Insert: 62
Insertion in a B-Tree

Insert: 62
Insertion in a B-Tree

Insert: 50

n = 2
Insertion in a B-Tree

Insert: 50

n = 2
Insertion in a B-Tree

Insert: 75
Insertion in a B-Tree

Insert: 75

n = 2
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion
Insertion: Primitives

- Inserting into a leaf node
- Splitting a leaf node
- Splitting an internal node
- Splitting root node
Inserting into a Leaf Node
Inserting into a Leaf Node
Inserting into a Leaf Node

| 54 | 57 | 58 | 60 | 62 | 58 |

58
Splitting a Leaf Node

54  66

54  57  58  60  62
Splitting a Leaf Node

54  66

54  57  58  60  62
Splitting a Leaf Node
Splitting a Leaf Node

59

54  66

54  57  58

60  61  62
Splitting a Leaf Node

54  59  66

54  57  58

60  61  62
Splitting an Internal Node

- [54, 59)
- [59, 66)
- [66, 74)
- ... 21 99 ...
- 59
Splitting an Internal Node

[40, 54, 66, 74, 84]

[54, 59)

[59, 66)

[66, 74)

[59, 99, ...]

[... 21 99 ...]
Splitting an Internal Node

[21, 66) → [54, 59) → [59, 66) → [66, 74)

[66, 99) → [66, 74) → [59, 66) → [54, 59) → [21, 66)
Splitting the Root

[54, 59)  [59, 66)  [66, 74)
Splitting the Root

[54, 59)  [59, 66)  [66, 74)
Splitting the Root

- Root node with value 66
- Subtrees with nodes 40, 54, 59 and 74, 84
- Intervals [54, 59), [59, 66), [66, 74)
Deletion
Deletion

redistribute
Deletion
Deletion - II
Deletion - II
Deletion - II
Deletion - II
Deletion - II
Deletion - II

Not needed

merge
Deletion - II
Deletion: Primitives

- Delete key from a leaf
- Redistribute keys between sibling leaves
- Merge a leaf into its sibling
- Redistribute keys between two sibling internal nodes
- Merge an internal node into its sibling
Merge Leaf into Sibling
Merge Leaf into Sibling
Merge Leaf into Sibling

72

54 58 64 68 75

... 67 85

Diagram showing the process of merging leaves into a sibling node in a data structure.
Merge Leaf into Sibling

72

54 58 64 68 75

... 85
Merge Internal Node into Sibling

41  48  52

[52, 59)  [59, 63)
Merge Internal Node into Sibling

\[
\begin{array}{cc}
\text{...} & 59 \\
\text{...} & \\
\end{array}
\]

\[
\begin{array}{cccc}
41 & 48 & 52 & 59 \\
\end{array}
\]

\[
\begin{array}{cccc}
52 & 59 & 63 & \\
\end{array}
\]

\[
\begin{array}{cccc}
[52, 59) & \\
\end{array}
\]

\[
\begin{array}{cccc}
[59, 63) & \\
\end{array}
\]
B-Tree Roadmap

- B-Tree
  - Recap
  - Insertion (recap)
  - Deletion
- Construction
- Efficiency
- B-Tree variants
- Hash-based Indexes
Question

How does insertion-based construction perform?
B-Tree Construction

Sort

48  57  41  15  75  21  62  34  81  11  97  13
B-Tree Construction

Scan
B-Tree Construction

Scan
B-Tree Construction

Why is sort-based construction better than insertion-based one?
Cost of B-Tree Operations

- Height of B-Tree: $H$
- Assume no duplicates
- Question: what is the random I/O cost of:
  - Insertion:
  - Deletion:
  - Equality search:
  - Range Search:
Height of B-Tree

- Number of keys: $N$
- B-Tree parameter: $n$

$$\text{Height} \approx \log_n N = \frac{\log N}{\log n}$$

In practice: 2-3 levels
Question: How do you pick parameter n?

1. Ignore inserts and deletes
2. Optimize for equality searches
3. Assume no duplicates
Roadmap

- B-Tree
- B-Tree variants
  - Sparse Index
  - Duplicate Keys
- Hash-based Indexes
Roadmap

- B-Tree
- B-Tree variants
- Hash-based Indexes
  - Static Hash Table
  - Extensible Hash Table
  - Linear Hash Table
Hash-Based Indexes

- Adaptations of main memory hash tables
- Support equality searches
- No range searches
Indexing Problem (recap)

Index Keys

\[ A = \text{val} \]

Record pointers
Main Memory Hash Table

\[ h(key) = key \mod 8 \]
Adapting to disk

- 1 Hash Bucket = 1 Block
  - All keys that hash to bucket stored in the block
  - Intuition: keys in a bucket usually accessed together
  - No need for linked lists of keys …
Adapting to Disk

How do we handle this?
Adapting to disk

1 Hash Bucket = 1 Block

- All keys that hash to bucket stored in the block
- Intuition: keys in a bucket usually accessed together
- No need for linked lists of keys …
- … but need linked list of blocks (overflow blocks)
Adapting to Disk
Adapting to disk

- Bucket Id → Disk Address mapping
  - Contiguous blocks
  - Store mapping in main memory
    - Too large?
    - Dynamic → Linear and Extensible hash tables
Beware of claims that assume 1 I/O for hash tables and 3 I/Os for B-Tree!!