Indexing: Part III

CPS 216
Advanced Database Systems

Announcements (February 15)

- Homework #1 graded
  - Verify your grades on Blackboard
- Homework #2 assigned today
  - Due in 2½ weeks
- Reading assignments for this and next week
  - “The” query processing survey by Graefe
  - Due next Wednesday
- Midterm and course project proposal in 3½ weeks

Static hashing

key $\rightarrow$ hash function $h$ $\rightarrow$ bucket number

bucket 0
bucket 1
bucket i
bucket overflow
bucket $\text{N-1}$

With records or record pointers

$\text{h}(k) = j$ ...

What if a bucket is full?

Does it make sense to use a hash-based index as a sparse index on a sorted table?
Performance of static hashing

- Depends on the quality of the hash function!
  - Best (hopefully average) case: one I/O!
  - Worst case: all keys hashed into one bucket!
  - See Knuth vol. 3 for good hash functions
- Rule of thumb: keep utilization at 50%-80%
- How do we cope with growth?
  - Extensible hashing
  - Linear hashing

Extensible hashing (TODS 1979)

- Idea 1: use $i$ bits of output by hash function and dynamically increase $i$ as needed
  
  $b(k) = 0101$

- Problem: $++i$ = double the number of buckets!
- Idea 2: use a directory
  - Just double the directory size
  - Many directory entries can point to the same bucket
  - Only split overflowed buckets
  - “One more level of indirection solves everything!”

Extensible hashing example (slide 1)

- Insert $k$ with $b(k) = 0101$

- Bucket too full?
  - $++$ local depth, split bucket, and $++$ global depth (double the directory size) if necessary
  - Allowing some overflow is fine too
Extensible hashing example (slide 2)

- Insert 1110, 0000

- Split again
  - No directory doubling this time

Extensible hashing example (slide 3)

- Insert 0001

Extensible hashing example (slide 4)

Delete is just the reverse:
- If bucket is too empty, merge with sibling bucket,
  - -- local depth,
  - if possible, -- global depth
  - and half the directory
Summary of extensible hashing

- **Pros**
  - Handles growing files
  - No full reorganization
- **Cons**

Linear hashing \((VLDB 1980)\)

- Grow only when utilization exceeds a given threshold
- No extra indirection
  - Some extra math to figure out the right bucket

\[ i = 1 \quad \text{Number of bits in use} = \left\lfloor \log_2 n \right\rfloor \]
\[ n = 2 \quad \text{Number of primary buckets} \]

Linear hashing example (slide 2)

- Grows linearly (hence the name)
- Always split the \((n - 2^{\lfloor \log_2 n \rfloor})\)-th bucket (0-based index)
  - Intuitively, the first bucket with the lowest depth
  - Not necessarily the bucket being inserted into!
Linear hashing example (slide 3)

Insert 1110
Threshold exceeded; grow!

Insertion table:

<table>
<thead>
<tr>
<th>i</th>
<th>n</th>
<th>Bucket 00 01 10 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>0000 0001 1010 1111</td>
</tr>
</tbody>
</table>

Threshold exceeded: grow!

\[ i = 2 \]
\[ n = 4 \]

Linear hashing example (slide 4)

- Look up 1110
  - Bucket 110 (6-th bucket) is not here
  - Then look in the \( (6 - 2^{\log_2 n}) \)-th bucket (= 2nd)

Look-up table:

<table>
<thead>
<tr>
<th>i</th>
<th>n</th>
<th>Bucket 00 01 10 11 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>0000 0001 1010 1111 1100</td>
</tr>
</tbody>
</table>

Threshold exceeded: grow!

\[ i = 3 \]
\[ n = 5 \]

Summary of linear hashing

- **Pros**
  - Handles growing files
  - No full reorganization

- **Cons**
Hashing versus B-trees

- Hashing is faster on average, but the worst case can be really bad
- B-trees provide performance guarantees, and they are not that tall in practice
- Hashing destroys order!
- B-trees provide order and support range queries