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

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
- 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 $h(k) = 0101$
- 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)
- Insert 0001
  - 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
  - One more level of indirection
  - Directory size still doubles
  - Sometimes doubling is not enough!

Linear hashing (VLDB 1980)
- Grow only when utilization exceeds a given threshold
- No extra indirection
  - Some extra math to figure out the right bucket
  - Intuitively, the first bucket with the lowest depth
  - Not necessarily the bucket being inserted into!

Linear hashing example (slide 2)
- Grows linearly (hence the name)
- Always split the \((n - 2^{\lceil \log n \rceil})\)-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!

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

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>1010</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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)

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

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>1010</th>
<th>1111</th>
<th>1100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of linear hashing

- **Pros**
  - Handles growing files
  - No full reorganization
  - No extra level of indirection

- **Cons**
  - Still has overflow chains
  - May not be able to split an overflow chain right away because buckets must be split in sequence

<table>
<thead>
<tr>
<th>empty</th>
<th>empty</th>
<th>empty</th>
<th>full</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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