CS222: Principles of Data Management

Notes #8

Linear Hashing

Instructor: Chen Li
Linear Hashing

- This is another dynamic hashing scheme, an alternative to Extendible Hashing.
- LH handles the problem of long overflow chains without using a directory, and handles duplicates.

**Idea:** Use a family of hash functions $h_0, h_1, h_2, ...$

- $h_i(key) = h(key) \mod (2^i N)$; $N = \text{initial # buckets}$
- $h$ is some hash function (range is not just 0 to N-1)
- If $N = 2^{d_0}$, for some $d_0$, $h_i$ consists of applying $h$ and looking at the last $d_i$ bits, where $d_i = d_0 + i$.
- $h_{i+1}$ doubles range of $h_i$ (≈directory doubling)

*Ex:* $d_0 = 2$ so $N = 4$
Linear Hashing (Contd.)

- Directory avoided in LH by using overflow pages, and choosing bucket to split round-robin.
  - Splitting proceeds in `rounds`. Round ends when all $N_R$ initial (for round $R$) buckets are split. Buckets in 0 to Next-1 have been split; Next to $N_R$ have yet to be split.
  - Current round number is called Level.

**Search:** To find bucket for data entry $r$, find $h_{\text{Level}}(r)$:
  - If $h_{\text{Level}}(r)$ in range `Next to $N_R$`, then $r$ belongs here.
  - Else, $r$ could belong to bucket $h_{\text{Level}}(r)$ or to bucket $h_{\text{Level}}(r) + N_R$; must apply $h_{\text{Level}+1}(r)$ to find out which.
Overview of LH File

- In the middle of a round.

Buckets that existed at the beginning of this round:
this is the range of \( h_{\text{Level}} \)

Bucket to be split

Next

Buckets split in this round:
If \( h_{\text{Level}} \) (search key value) is in this range, must use \( h_{\text{Level}+1} \) (search key value) to decide if entry is in `split image' bucket.

`Split image' buckets:
created (through splitting of other buckets) in this round.
Linear Hashing (Contd.)

- **Insert**: Find bucket by applying $h_{\text{Level}} / h_{\text{Level}+1}$:
  - If bucket to insert into is full:
    - Add overflow page and insert data entry.
    - *(Maybe)* Split *Next* bucket and increment *Next*.

- Can choose any criterion to ‘trigger’ a split.
- Since buckets are split round-robin, long overflow chains don’t develop! *(See why?)*
- Doubling of directory in Extendible Hashing is similar; switching of hash functions is *implicit* in how the # of bits examined in EH is increased.
Example of Linear Hashing

- On split, $h_{Level+1}$ is used to re-distribute entries.

<table>
<thead>
<tr>
<th>Level=0, N=4</th>
<th>Insert 43* (43 = 101011)</th>
<th>Level=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h 1</td>
<td>h 0</td>
<td>PRIMARY PAGES</td>
</tr>
<tr>
<td>000</td>
<td>00</td>
<td>32<em>44</em>36*</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>9<em>25</em>5*</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
<td>14<em>18</em>10<em>30</em></td>
</tr>
<tr>
<td>011</td>
<td>11</td>
<td>31<em>35</em>7<em>11</em></td>
</tr>
</tbody>
</table>

*(h_i info is shown only for illustration....!)* *(Actual contents of linear hashed file)*
Example: End of a Round

Level=0

PRIMARY PAGES

h_1 | h_0
---|---
000 | 00
  | 32*

001 | 01
  | 9* 25*

010 | 10
  | 66* 18* 10* 34*

011 | 11
  | 31* 35* 7* 11*

Next=3

OVERFLOW PAGES

h_1 | h_0
---|---
000 | 00
  | 32*

001 | 01
  | 9* 25*

010 | 10
  | 66* 18* 10* 34*

011 | 11
  | 43* 35* 11*

100 | 00
  | 44* 36*

101 | 11
  | 5* 37* 29*

110 | 10
  | 14* 30* 22*

111 | 11
  | 31* 7*

Next=0

Level=1

PRIMARY PAGES

h_1 | h_0
---|---
000 | 00
  | 32*

001 | 01
  | 9* 25*

010 | 10
  | 66* 18* 10* 34*

011 | 11
  | 43* 35* 11*

100 | 00
  | 44* 36*

101 | 11
  | 5* 37* 29*

110 | 10
  | 14* 30* 22*

111 | 11
  | 31* 7*

Insert 50* (50 = 110010)

(50 = 110010)
<table>
<thead>
<tr>
<th></th>
<th>(a) Scan</th>
<th>(b) Equality</th>
<th>(c) Range</th>
<th>(d) Insert</th>
<th>(e) Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Heap</td>
<td>BD</td>
<td>0.5BD</td>
<td>BD</td>
<td>2D</td>
<td>Search + D</td>
</tr>
<tr>
<td>(2) Sorted</td>
<td>BD</td>
<td>Dlog 2B</td>
<td>D(log 2 B + # pgs with match recs)</td>
<td>Search + BD</td>
<td>Search + BD</td>
</tr>
<tr>
<td>(3) Clustered</td>
<td>1.5BD</td>
<td>Dlog F 1.5B</td>
<td>D(log F 1.5B + # pgs w. match recs)</td>
<td>Search + D</td>
<td>Search + D</td>
</tr>
<tr>
<td>(4) Unclust. Tree index</td>
<td>BD(R+0.15)</td>
<td>D(1 + log F 0.15B)</td>
<td>D(log F 0.15B + # pgs w. match recs)</td>
<td>Search + 2D</td>
<td>Search + 2D</td>
</tr>
<tr>
<td>(5) Unclust. Hash index</td>
<td>BD(R+0.125)</td>
<td>2D</td>
<td>BD</td>
<td>Search + 2D</td>
<td>Search + 2D</td>
</tr>
</tbody>
</table>

Several assumptions underlie these (rough) estimates!
Summary (Cont’d.)

- Linear Hashing avoids directory by splitting buckets round-robin and (still) using overflow pages.
  - Overflow chains not likely to be long.
  - Duplicates handled more easily.
  - Space utilization could be lower than Extendible Hashing, since splits are not concentrated on `dense’ data areas.
    - Can tune LH criterion for triggering splits to trade-off slightly longer chains for better space utilization.

- Note: For hash-based indexes, a skewed data distribution is one in which the hash values of data entries are not uniformly distributed!
Summary

- Hash-based indexes: best for equality searches, cannot support range searches (i.e., not efficiently).
- Static Hashing can lead to long overflow chains.
- Extendible Hashing avoids overflow pages by splitting a full bucket when a new data entry is added to it. *(Duplicates may require overflow pages.)*
  - Directory to keep track of buckets; doubles periodically.
  - Directory can get large with skewed data; additional I/O if it does not fit in main memory.