Database Systems
(資料庫系統)
November 26/28, 2007
Lecture #9

Announcement

- Hand in your assignment #4 on Wed.
African Poverb

Every morning, a zebra wakes up, it knows that it must run faster than the fastest lion, or it will be eaten; every morning, a lion wakes up, it knows that it must run faster than the slowest zebra, or it will starve to death; it does not matter whether you are a zebra or a lion, when the sun comes up, you better run fast.

Hash-Based Indexing

Chapter 11
Introduction

- Hash-based indexes are best for equality selections. Cannot support range searches.
  - Equality selections are useful for natural join operations.

Review: Tree Index

- ISAM vs. B+ trees.
  - What is the main difference between them?
    - Static vs. dynamic structure
  - What is the tradeoff between them?
    - Graceful table growth/shrink vs. concurrency performance
Similar tradeoff for hashing

• Possible to exploit the tradeoff between dynamic structure for better performance?

• Static hashing technique

• Two dynamic hashing techniques
  – Extendible Hashing
  – Linear Hashing

Basic Static Hashing

• # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.

• \( h(k) \mod N = \) bucket to which data entry with key \( k \) belongs. (\( N = \) # of buckets)
  – \( N = 2^i \) (or the bucket number is the i-least significant bits)

\[
\begin{align*}
  h(key) \mod N \\
  key \\
  0 \rightarrow 0^* \\
  1 \rightarrow 3^*, 5^* \\
  2 \rightarrow 4^*, 6^* \\
  3 \rightarrow 9^* \\
  N-1 \rightarrow 11^* \\
  \text{Primary bucket pages} \\
  \text{Overflow pages}
\end{align*}
\]
Static Hashing (Contd.)

- Buckets contain data entries.
- Hash function takes search key field of record r and distribute values over range 0 ... N-1.
  - $h(key) = (a * key + b)$ usually works well.
  - $a$ and $b$ are constants; lots known about how to tune $h$.
- Cost for insertion/delete/search: 2/2/1 disk page I/Os (no overflow chains).

What is the performance problem for static hashing?

- Long overflow chains can develop and degrade performance.
  - Why poor performance? Scan through overflow chains linearly.
- How to fix this problem?
  - Extendible and Linear Hashing: Dynamic techniques to fix this problem.
Extendible Hashing

- Simple Solution – remove overflow chain.
- When bucket (primary page) becomes full, ...
  - Re-organize file by adding (doubling) buckets.
- What is the cost concern in doubling buckets?
  - High cost: rehash all entries - reading and writing all pages is expensive!

\[
\begin{align*}
&h'(key) = h(key) \text{ mod } N \\
&\text{key} \\
&h \\
&0 \rightarrow 0^* \\
&1 \rightarrow 2^*, 5^* \\
&2 \rightarrow 4^* \\
&\vdots \\
&N-1 \rightarrow N^* \\
&N \rightarrow N+1 \\
&N+1 \rightarrow \ldots \\
&2N-1 \\
\end{align*}
\]

How to minimize the rehashing cost?

- Use another level of indirection
  - Directory of pointers to buckets
- Insert 0 (00)
  - Double buckets only double the directory (no rehashing)
  - Split just the bucket that overflowed!
- What is cost here?

\[
\begin{align*}
\text{Directory} \\
00 &\rightarrow 4^* \ 12^* \ 32^* \ 8^* \\
01 &\rightarrow 1^* \ 5^* \ 21^* \ 13^* \\
10 &\rightarrow 10^* \\
11 &\rightarrow 15^* \ 7^* \ 39^* \\
\end{align*}
\]

\[
\begin{align*}
\text{Doubled Directory} \\
\end{align*}
\]

\[
\begin{align*}
\text{Bucket A} \\
\text{Bucket B} \\
\text{Bucket C} \\
\text{Bucket D} \\
\text{Bucket A1} \\
\end{align*}
\]
Extendible Hashing

- Directory much smaller than file, so doubling much cheaper. Only one page of data entries is split.
- How to adjust the hash function?
  - Before doubling directory
    - $h(r) \rightarrow 0..N-1$ buckets.
  - After doubling directory?
    - $h(r) \rightarrow 0..2N-1$ (the range is doubled)

Example

- Directory is array of size 4.
- What info this hash data structure maintains?
  - Need to know current #bits used in the hash function
    - Global Depth
  - May need to know #bits used to hash a specific bucket
    - Local Depth
  - Can global depth < local depth?
**Insert 20 (10100): Double Directory**

- Increment global depth
- Rehash bucket A
- Increment local depth, why track local depth?

**Insert 9 (1001): only Split Bucket**

- Rehash bucket B
- Increment local depth
Points to Note

- What is the global depth of directory?
  - Max # of bits needed to tell which bucket an entry belongs to.
- What is the local depth of a bucket?
  - # of bits used to determine if an entry belongs to this bucket.
- When does bucket split cause directory doubling?
  - Before insert, bucket is full & local depth = global depth.
- How to efficiently double the directory?
  - Directory is doubled by copying it over and ‘fixing’ pointer to split image page.
  - Note that you can do this only by using the least significant bits in the directory.

Directory Doubling

Why use least significant bits in directory?
⇔ Allows for doubling via copying!

Least Significant vs. Most Significant

Split buckets
Directory Copying via Insertion 20 (10100)

Comments on Extendible Hashing

- If directory fits in memory, equality search answered with one disk access; else two.
- What is the problem with extendible hashing?
  - Consider if the distribution of hash values is skewed (concentrates on a few buckets)
  - Directory can grow large.
- Can you come up with one insertion leading to multiple splits?
Skewed data distribution (multiple splits)

- Assume each bucket holds two data entries
- Insert 2 (binary 10) – how many times of DIR doubling?
- Insert 16 (binary 10000) – how many times of DIR doubling?
- How to solve this data skew problem?

Insert 2 (10)
Comments on Extendible Hashing

- Delete: If removal of data entry makes bucket empty, can be merged with `split image`. If each directory element points to same bucket as its split image, can halve directory.
Delete 10*

Delete 15*, 7*, 19*
Linear Hashing (LH)

- Another dynamic hashing scheme, as an alternative to Extendible Hashing.
- What are problems in static/extendible hashing?
  - Static hashing: long overflow chains
  - Extendible hashing: data skew causing large directory
- Is it possible to come up with a more balanced solution?
  - Shorter overflow chains than static hashing
  - No need for directory in extendible hashing
    - How do you get rid of directory (indirection)?

Linear Hashing (LH)

- Basic Idea:
  - Pages are split when overflows occur – but not necessarily the page with the overflow.
  - Splitting occurs in turn, in a round robin fashion.
    - one by one from the first bucket to the last bucket.

- Use a family of hash functions \( h_0, h_2, h_3, \ldots \)
  - Each function’s range is twice that of its predecessor.
  - When all the pages at one level (the current hash function) have been split, a new level is applied.
  - Splitting occurs gradually
  - Primary pages are allocated in order & consecutively.
Linear Hashing Verbal Algorithm

• Initial Stage.
  – The initial level distributes entries into $N_0$ buckets.
  – Call the hash function to perform this $h_0$.

• Splitting buckets.
  – If a bucket overflows its primary page is chained to an overflow page (same as in static hashing).
  – Also when a bucket overflows, some bucket is split.
    • The first bucket to be split is the first bucket in the file (not necessarily the bucket that overflows).
    • The next bucket to be split is the second bucket in the file … and so on until the $N$th. has been split.
    • When buckets are split their entries (including those in overflow pages) are distributed using $h_1$.
  – To access split buckets the next level hash function ($h_1$) is applied.
  – $h_1$ maps entries to $2N_0$ (or $N_1$) buckets.
### Linear Hashing Example

- Initially, the index level equal to 0 and $N_0$ equals 4 (three entries fit on a page).
- $h_0$ range = 4 buckets
- Note that `next` indicates which bucket is to split next. (Round Robin)
- Now consider what happens when 9 (1001) is inserted (which will not fit in the second bucket).

<table>
<thead>
<tr>
<th>$h_0$</th>
<th></th>
<th>next</th>
<th>64</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>1</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>31</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

### Insert 9 (1001)

- $h_0$ next
  - $h_0$ range = 4 buckets
- Note that `next` indicates which bucket is to split next. (Round Robin)
- Now consider what happens when 9 (1001) is inserted (which will not fit in the second bucket).

<table>
<thead>
<tr>
<th>$h_0$</th>
<th></th>
<th>next</th>
<th>64</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>1</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>31</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

| $h_0$ |   | next | 1  | 17 | 5  | 9  |
|-------|---|------|----|----|----|
| 00    |   |      |    |    |    |
| 01    |   |      |    |    |    |
| 10    |   |      |    |    |    |
| 11    |   |      |    |    |    |
Linear Hashing Example 2

<table>
<thead>
<tr>
<th>Page</th>
<th>next</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(000)H₁</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>(01)H₀</td>
<td>1 17 5</td>
<td></td>
</tr>
<tr>
<td>(10)H₀</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>(11)H₀</td>
<td>31 15</td>
<td></td>
</tr>
<tr>
<td>(100)H₁</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

- The page indicated by next is split (the first one).
- Next is incremented.

- An overflow page is chained to the primary page to contain the inserted value.
- Note that the split page is not necessary the overflow page – round robin.
- If h₀ maps a value from zero to next – 1 (just the first page in this case), h₁ must be used to insert the new entry.
- Note how the new page falls naturally into the sequence as the fifth page.

Insert 8 (1000), 7 (111), 18 (10010), 14 (1100)

<table>
<thead>
<tr>
<th>Page</th>
<th>next</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(000)H₁</td>
<td>64 8</td>
<td></td>
</tr>
<tr>
<td>(01)H₀</td>
<td>1 17 5</td>
<td></td>
</tr>
<tr>
<td>(10)H₀</td>
<td>6 18</td>
<td></td>
</tr>
<tr>
<td>(11)H₀</td>
<td>31 15 7</td>
<td></td>
</tr>
<tr>
<td>(100)H₁</td>
<td>36 14</td>
<td></td>
</tr>
<tr>
<td>Hash Function</td>
<td>Before Insert 11 (1011)</td>
<td>After Insert 11 (1011)</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>(000)H₁</td>
<td>64 8</td>
<td>64 8</td>
</tr>
<tr>
<td>(01)H₀</td>
<td>1 17 5</td>
<td>1 17 9</td>
</tr>
<tr>
<td>(10)H₀</td>
<td>6 18</td>
<td>6 18</td>
</tr>
<tr>
<td>(11)H₀</td>
<td>31 15 7</td>
<td>31 15 7</td>
</tr>
<tr>
<td>(100)H₁</td>
<td>36 14</td>
<td>36 14</td>
</tr>
<tr>
<td>(101)H₁</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Which hash function (h₀ or h₁) for searching 9?

How about searching for 18?
Linear Hashing

- Insert 32, 16, 2
- Insert 10, 13, 23
- After the 2nd. split the base level is 1 ($N_1 = 8$), use $h_1$.
- Subsequent splits will use $h_2$ for inserts between the first bucket and next-1.

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$h_1$</th>
<th>$h_1$</th>
<th>$h_1$</th>
<th>$h_1$</th>
<th>$- $</th>
<th>$- $</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>next$_1$</td>
<td>64</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>$h_1$</td>
<td>$h_0$</td>
<td>$h_0$</td>
<td>next$_1$</td>
<td>10</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>$h_0$</td>
<td>$h_0$</td>
<td>next$_2$</td>
<td>11</td>
<td>15</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>$h_1$</td>
<td>$h_1$</td>
<td>$h_1$</td>
<td>$h_1$</td>
<td>$h_1$</td>
<td>$h_1$</td>
<td>$h_1$</td>
</tr>
<tr>
<td>36</td>
<td>5</td>
<td>13</td>
<td>6</td>
<td>14</td>
<td>31</td>
<td>15</td>
</tr>
</tbody>
</table>

Notes for Linear Hashing

- Why linear hashing does not need a directory but extendible hashing needs one?
  - Consider finding i-th bucket
  - New buckets are created in order (sequentially, i.e., round robin) in linear hashing
- Level progression:
  - Once all $Ni$ buckets of the current level (i) are split the hash function $h_i$ is replaced by $h_{i+1}$.
  - The splitting process starts again at the first bucket and $h_{i+2}$ is applied to find entries in split buckets.
LH Described as a Variant of EH

- Can you describe linear hashing using extendible hashing (EH)?
  - Begin with an EH index where directory has $N$ elements.
  - Use overflow pages, split buckets round-robin.
  - First split is at bucket 0. (Imagine directory being doubled at this point.)
    But elements $<1,N+1>$, $<2,N+2>$, ... are the same. So, need only create
directory element $N$, which differs from 0, now.
    - When bucket 1 splits, create directory element $N+1$, etc.
- So, directory can double gradually. Also, primary bucket pages
  are created in order. If they are allocated in sequence too (so
  that finding i’th is easy), we actually don’t need a directory!
Voila, LH.
The New World is Flat

• With computer and network, the playing field is being leveled across the world.
• There is no end to what can be done by whom
  – Before: manufacturing products (Made in Taiwan, Made in China)
  – Now: anything including service industry
• This is called outsourcing
• The rule of Capitalist economy:
  – Work goes to the place where it can be done cheapest and most efficient

Outsourcing Examples

• Customer services moving to call centers in India
  – Credit cards, banks, insurance, Microsoft, ...
• Radiologists in India and Australia
• E-tutoring
• Jetblue & homesourcing
• Tax preparation outsourcing
• MacDonald Flatfries and burgers
• Kenichi Ohmae (大前研一)
Outsourcing and ME in Taiwan?

• So anyone on this planet may be replaced by anyone on this planet for cheaper and more efficiency
  – Yes, for any jobs – high/low tech, manufacturing/service
  – Scaring? Consider massive amount of smart & cheap labors in China and India
• Oops ... this is happening NOW here in Taiwan!

• So one can compete or collaborate globally with anyone on this planet -> opportunity.