COS 597A:
Principles of
Database and Information Systems

Indexing, Part II

1

Dynamic hashing

- Have talked about static hash
  - Pick a hash function and bucket organization and keep it
  - Assume (hope) inserts/deletes balance out
  - Use overflow blocks as necessary
- What if database growing?
  - Overflow blocks may get too plentiful
  - Reorganize hash buckets to eliminate overflow buckets
    - Can't completely eliminate

2

Family of hash functions

- Static hashing:
  choose one good hash function h
  - What is "good"?

- Dynamic hashing:
  chose a family of good hash functions
  - $h_0, h_1, h_2, h_3, \ldots h_k$
  - h_{i+1} refines h_i :
    if $h_{i+1}(x) = h_{i+1}(y)$ then $h_i(x) = h_i(y)$

3

A particular hash function family

- Commonly used: integers mod 2^i
  - Easy: low order i bits
- Base hash function can be any h mapping hash field values to positive integers
- $h_0(x) = h(x) \bmod 2^b$ for a chosen b
  - 2^b buckets initially
- $h_i(x) = h(x) \bmod 2^{b+i}$
  - Double buckets each refinement
- *If x integer, h(x)= x sometimes used*
  ➤ What does this assume for h_0 to be *good*?

4

Specifics of dynamic hashing

- Conceptually double # buckets when reorganize
- Implementationally don't want to allocate space may not need
  - One bucket overflows, double all buckets? NO!
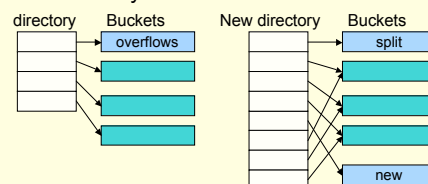
Solution?
Extendable hashing
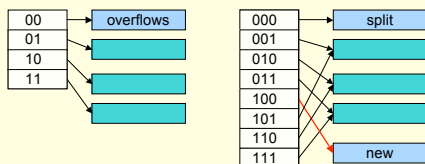 - Reorganize when and where need

5

Extendable hashing

- When a bucket overflows,
  - actually split that bucket in two
  - Conceptually split all buckets in two
- Use directory to achieve:

6

1

Extendable hashing details

- Indexing directory with $h_i(x) = h(x) \bmod 2^{b+i}$
- On overflow, index directory with
  $h_{i+1}(x) = h(x) \bmod 2^{b+i+1}$
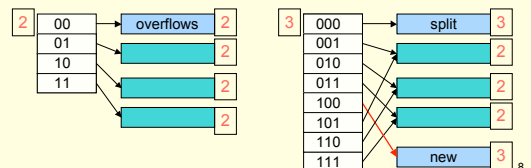- Directory size doubles
- Add one bucket

- What did we do?
  - Split overflowing bucket m
    - Allocate new bucket
  - Copy directory
  - Change pointer of directory entry $m+2^{b+i}$

Keep track of how many bits actually using
  - depth of directory: global depth
  - depth of each bucket: local depth (WHY KEEP?)


Rule of bucket splitting

- On bucket m overflow:
  - If depth(directory) > depth(bucket m)
    - Split bucket m into bucket m and bucket $m+2^{depth(m)}$
    - Update depth buckets m and $m+2^{depth(m)}$
    - Update pointers for all directory entries pointing to m

  - If depth(directory) = depth(bucket m)
    - Split bucket m into bucket m and bucket $m+2^{depth(m)}$
    - Update depth buckets m and $m+2^{depth(m)}$
    - Copy directory and update depth(directory)
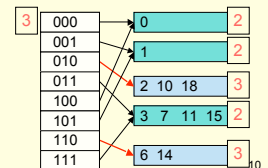   - Change pointer of directory entry $m+2^{depth(m)}$

Example

Buckets: max 4 keys and data per bucket
Start with 4 buckets: depth(directory)=2

Insert records with hash values $h(r)$ = 0, 1, 2, 3, 6, 10, 14, 7, 11, 15:
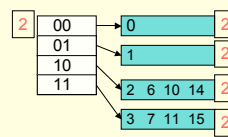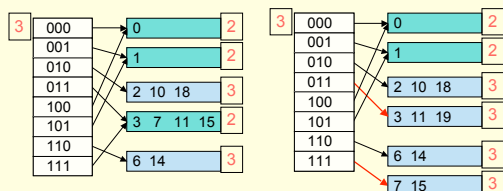
Then insert $h(r)$ = 18 bucket '10' overflows => split

Example continued

Buckets: max 4 keys and data per bucket

After inserted $h(r)$=18:

Then insert $h(r)$ = 19 bucket '11' overflows => split

---

Extendable hashing observations

- Splitting bucket does not always evenly distribute contents
  - $h_i(x)$ may equal $h_{i+1}(x)$, $h_{i+2}(x)$, …
- May need to split bucket several times
  - NOT: global depth – min(local depth) = 1
- Can accept some overflow blocks or split aggressively
- Almost no overflow blocks with good hash function and aggressive splitting.
- If $h(x) = h(y)$ always same bucket
  - cannot avoid overflow if too many of these!
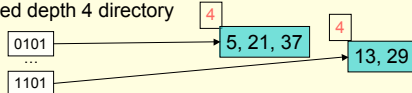
Example bad bucket overflow

Bucket:

2

5, 13, 21, 29
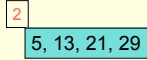
$h(key) \bmod 2^2 = 1$
$h(key) \bmod 2^3 = 5$
If add new entry with $h(key) = 37$ then $h(key) \bmod 2^3 = 5$
=>splitting once not enough
Need depth 4 directory

4

4

0101
...
1101

5, 21, 37

13, 29

13

Index Operation Costs

14

Extendable Hashing Costs

Assume: One block per bucket; no overflow blocks

- Look up: # blocks read = 1 + 1
  - Assumes directory on disk
- Insert without overflow
  = look-up cost + 1 to write block of bucket
- Insert with overflow - splitting once:
  = look-up cost + 1 to write block of original bucket
    + 1 to write block of new bucket
    + 2 * (# disk blocks of directory) to copy
    + 1 to change link for original matching directory entry
- Splitting once may not be enough

15

Extendable Hashing Costs

One block per bucket; use some overflow blocks

- Look up: add (# overflow blocks) worst case
- Insert without splitting: add 1 **if** add new overflow block
- Insert with splitting once:
  add (# overflow blocks) **always** to look-up cost
  add (# overflow blocks) to write cost worst case
  - must read overflow blocks to split
  - adding 1 new bucket (block), so end up with # overflow blocks within 1 of number had before

16

B+ tree costs: preliminaries

- height of B+ tree = length of path: root → leaf
  $\leq \lceil \log_{d+1}(N) \rceil + 1$
  - N is number of leaves of tree
  - d+1 is min fanout of interior nodes except root
  - + 1 is for root

- typically root kept in memory
  - keep as many levels of tree as can in memory
  - buffer replacement algorithm may do,
    or pin

17

B+ tree costs: What is N?

- B+ tree file organization:
  - each leaf holds records
    $N \geq \lceil (\text{ # records in file / # records fit in a block }) \rceil$
    $N \leq 2 * \lceil (\text{ # records in file / # records fit in a block }) \rceil$
    assuming no duplicate search key values

- B+ tree primary index on sequential file:
  - each leaf holds pointers to file blocks
    - can be sparse index
      - one key value (smallest) for each file block
    - (key value, pointer) pairs in leaves
      - assume fit between d and 2d in leaf
    $\lceil (\text{# blocks in file) / 2d)} \rceil \leq N \leq \lceil (\text{# blocks in file) / d)} \rceil$
    - assumes no key value spans multiple blocks

18

B+ tree costs: What is N?

- B+ tree secondary index:
  - each leaf holds pointers to block of pointers
    - indirection: pointers in point to records
    - must be dense
    - (key value, pointer) pairs in leaves
      - assume fit between d and 2d in leaf

  $N \leq \lceil (\text{\# key values in file}) / d \rceil$
  $N \geq \lceil (\text{\# key values in file}) / 2d \rceil$

19

B+ tree costs: retrieval

- retrieving single record
  # of blocks accessed =
  height of B+-tree
  + 1 for root if on disk
  $+ \begin{cases} 1 & \text{if leaves pt to records} \\ 2 & \text{if leaves pt to block of pointers to records} \end{cases}$

  $\leq \lceil \log_{d+1} (N) \rceil + 3$

- typical height?

20