COS 425:
Database and Information
Management Systems

Indexing files, 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 pages as necessary
- What if database growing?
  - Overflow pages 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, … 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!

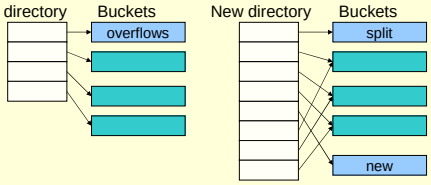
Solution? R&G text presents two versions:

- Extendible hashing
  - Reorganize when and where need
- Linear hashing
  - Reorganize when need but not where need
  - Reduces overflow buckets *on average*

5

Extendible hashing

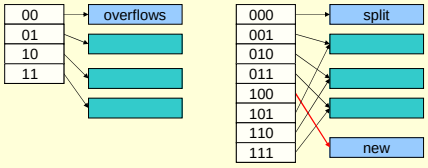
- When a bucket overflows,
  - actually split that bucket in two
  - Conceptually split all buckets in two
- Use directory to achieve:



6

Extendible hashing details

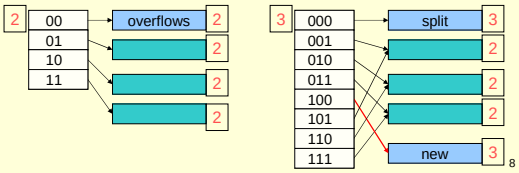
- Indexing directory with $h_i(x) = h(x) \bmod 2^{b+i}$
- On overflow, index directory with $h_{i+1}(x) = h(x) \bmod 2^{b+i+1}$
- Directory size doubles
- Add one bucket



7

- 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?)



8

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 $m+2^{depth(m)}$

9

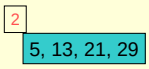
Extendible hashing observations

- 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 pages or split aggressively
- If h(x) = h(y) always same bucket
  - cannot avoid overflow if too many of these!

10

Example bad bucket overflow

Bucket:



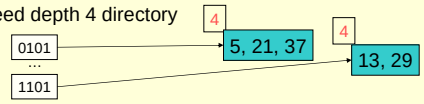
h(key) mod 2^2 = 1

h(key) mod 2^3 = 5

If add new entry with h(key)= 37 then h(key) mod 2^3 = 5

=>splitting once not enough

Need depth 4 directory



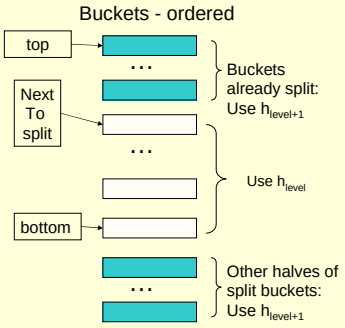
11

Linear Hashing

- Goal: get rid of directory of extendible
- Compromise:
  will tolerate overflow pages temporarily
- Idea:
  - Use same family of hash functions (mod 2^{b+i})
  - When bucket overflows split *some* bucket
  - Split buckets in order
  - Eventually bucket with overflow pages will get turn to split

12

Linear hashing details



- Have rounds of splitting: top to bottom
  - level
- Start new round when bottom bucket has split
- Next bucket splits when some criterion triggers
- Not shown: any bucket can have overflow pages

13

Linear hashing: more details

- Splitting criterion flexible
  - Basic: every time add overflow page
  - Alternate: every time bucket first overflows
- No directory => hash indexes buckets directly
  - Sequentially stored
- # buckets at level 0 need not be a power of 2
  - Values *top, bottom* suffice
  - h_i must be consistent with number buckets at level=i
- *Is true* every bucket at beginning of round has split by end of round
- *Is NOT true* no overflow at end of round

14

Board Example

15

Compare: Extendible vs Linear

- Extendible
  - Split actual bucket need to split
  - Need directory to tell where new bucket is
  - Duplicate directory cheaper
- Linear
  - No directory
  - Must keep buckets linearly ordered
    - Array access: calculate bucket location from hash
  - Relying on aggregation of splits over time to reduce overflow pages

16

Costs

- Look up: # pages accessed
  - Extendible hash: = 1 + 1 + (# overflow pages)
    - Assumes directory on disk
    - Almost no overflow pages with good hash function and aggressive splitting.
  - Linear hash: = 1 + (# overflow pages)
- Insert with overflow:
  - Extendible
    - Copy directory (# disk pages?)
    - Splitting once may not be enough
  - Linear
    - Follow overflow links
    - Split one bucket (assuming criterion met)

17