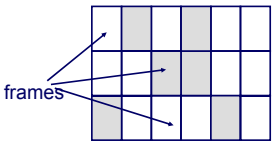
15-721 Database Management Systems

Buffer Management: DBMin

Instructor: Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Review

- DB accesses are page-oriented
- Need to cache DBMS disk pages
- **Buffer pool:** a set of page frames, each of which can hold a disk page.



Interface

- Hash table maps pageID to BP index entries
- getpage(pageNo) – returns memory address
  - Check buffer pool for page
  - If not found, get from disk
  - Fix page in buffer pool
  - Note the reference
  - Return address of page

Interface (cont'd)

- unfixpage(pageNo) – decrements fix count
- flushpage(pageNo) – force page to disk

Chou and DeWitt - Outline

- Review of Algorithms
  - Domain separation (Reiter)
  - Extensions to domain separation algorithm
  - "new" algorithm
  - Hot-set algorithm
- DBMIN - ideas and algorithms
- Experiments

Domain Separation [Reiter76]

- Classify pages as types
- Each type has a domain of buffers
- "borrow" page from one domain to another
- LRU within domain
- Example: B^+-tree index
  - One domain per index level
  - One domain per leaf/data pages

Domain Separation: Problems

- Problems ?

Domain Separation: Problems

- Problems with this approach
  - Static domains (relative importance depends on query)
  - Doesn't prevent interference among users
  - No load control => thrashing may occur

Domain Separation: Extensions

- Group LRU (GLRU) [Nybe84]
  - Fixed priority ranking for domains to find free pages
  - Search for free buffers in lowest priority group
- Working-set-like partitioning [Effe84]
  - Dynamically vary domain size
  - Do not replace pages in domain i referenced in last ti references

“New” algorithm [Kaplan80]

- Page priority a property of the *relation*
- Therefore, each relation needs a working set
- INGRES proposal: “new” algorithm
  - Each active relation is assigned part of buffer pool
  - Resident sets linked in priority order
  - Global free list on top
  - Page fault: search for free page via priority chain
  - Use MRU for resident sets (but keep >=1 active buffer)

“New” algorithm: Problems

- Problems ?

“New” algorithm: Problems

- How to determine priority?
- MRU not always good
- Costly search under high loads
- Not multi-user (hard to determine priority)
- Didn’t improve LRU performance

Hot Set [Sacc82]

- Query behavior model
- *hot set*: set of pages over which there is looping behavior
- hot set in memory ⇒ efficient query processing
- #page faults vs. size of buffers in partitions
  - Discontinuities: *hot points*

Hot Set: key ideas

- Give query |hot set| pages
- Allow ≤ 1 deficient query to execute
- Query optimizer determines hot set size
- LRU within each partition
- New query
  - Allowed in if hot set size <=free space
  - New buffer # pages = hot set size

Hot Set (cont'd)

- Problems ?

Hot Set (cont'd)

- Problems
  - LRU not always fast => allocate more memory!
    - MRU better for looping
  - Over-allocates pages for some phases of query
    - => under-utilized memory

Chou and DeWitt - Outline

- Review of Algorithms
  - Domain separation (Reiter)
  - ...
- DBMIN - ideas and algorithms
- Experiments

DBMIN [Chou & DeWitt 85]

- Based on "Query Locality Set Model"
- DBMS support a limited set of operations
- Reference patterns exhibited are predictable

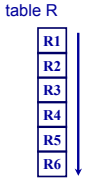
- Decompose complex patterns into simple
- Identify locality sets

Which patterns?

- Sequential (+ variations)
- Random
- Hierarchical

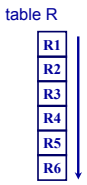
Sequential Patterns

- Straight sequential (SS)
  - File scan
  - #pages?
  - Replacement algorithm?



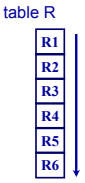
Sequential Patterns (cont.)

- Straight sequential (SS)
  - File scan
  - Need one page
  - Replacement algorithm?



Sequential Patterns (cont.)

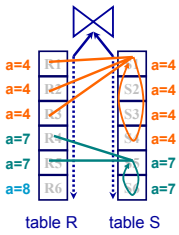
- Straight sequential (SS)
  - File scan
  - Need one page
  - Replaced with next one



Sequential Patterns (cont.)

Clustered sequential (CS)

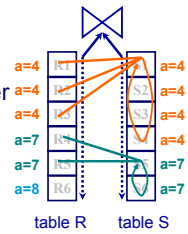
- Like inner S for merge-join (sequential w/ backup)
- Join condition: R.a=S.a
- # of pages?
- Replacement algorithm?



Sequential Patterns (cont.)

Clustered sequential (CS)

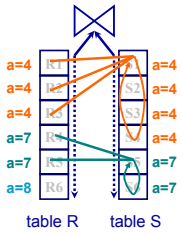
- Like inner S for merge-join (sequential w/ backup)
- # of pages in largest cluster
- Replacement algorithm?



Sequential Patterns (cont.)

Clustered sequential (CS)

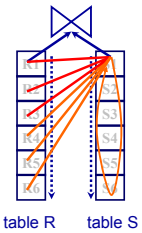
- Like inner S for merge-join (sequential w/ backup)
- # of pages in largest cluster
- FIFO or LRU



Sequential Patterns (cont.)

Looping sequential (LS)

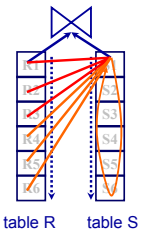
- Like inner S for nested-loop-join
- # of pages?
- Replacement algorithm?



Sequential Patterns (cont.)

Looping sequential (LS)

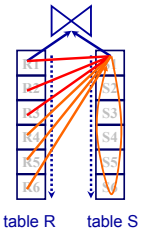
- Like inner S for nested-loop-join
- As many pages as possible
- Replacement algorithm?



Sequential Patterns (cont.)

Looping sequential (LS)

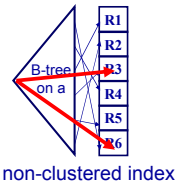
- Like inner S for nested-loop-join
- As many pages as possible
- MRU



Random Patterns

Independent Random (IR)

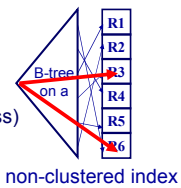
- Non-clustered index scan
- # of pages?
- Replacement algorithm?



Random Patterns (cont.)

Independent Random (IR)

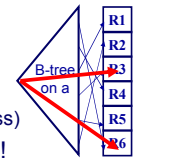
- Non-clustered index scan
- One page
  (assuming low prob. of reaccess)
- Replacement algorithm?



Random Patterns (cont.)

Independent Random (IR)

- Non-clustered index scan
- One page
  (assuming low prob. of reaccess)
- Any replacement algorithm!

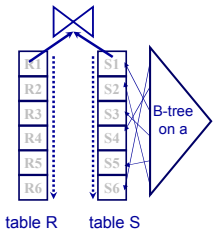


non-clustered index

Random Patterns (cont.)

Clustered Random (CR)

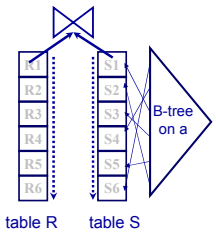
- Inner, non-clustered index on join column
- # of pages?
- Replacement algorithm?



Random Patterns (cont.)

Clustered Random (CR)

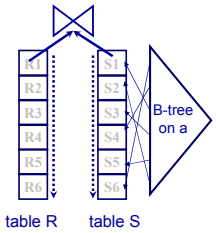
- Inner, non-clustered index on join column
- # of *records* in largest cluster
- Replacement algorithm?



Random Patterns (cont.)

Clustered Random (CR)

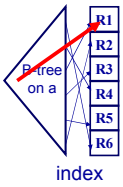
- Inner, non-clustered index on join column
- # of *records* in largest cluster
- as in CS (FIFO or LRU)



Hierarchical Patterns

Straight Hierarchical (SH)

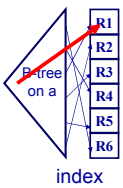
- Access index pages ONCE (retrieve a single tuple)
- # of pages?
- Replacement algorithm?



Hierarchical Patterns (cont.)

Straight Hierarchical (SH)

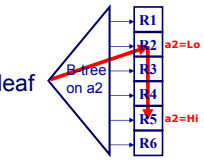
- Access index pages ONCE (retrieve a single tuple)
- Like SS



Hierarchical Patterns (cont.)

Hierarchical w/
straight/clustered sequential
(H/SS or H/CS)

- Hierarchical w/ SS or CS leaf scan
- Like SS/CS



e.g., clustered index

Hierarchical Patterns (cont.)

Looping Hierarchical (LS)

- When inner index in join is repeatedly accessed
- LIFO need to keep root

Chou and DeWitt - Outline

- Review of Algorithms
  - Domain separation (Reiter)
  - ...
- DBMIN - ideas and **algorithms**
- Experiments

DBMIN policy

- Buffers allocated on a *per-file-instance* basis
- Different BP for each active instance of file i
  - Set of pages of a file instance = *locality set* (lset)
  - Locality sets are independently managed
- Each page in buffer belongs to at most 1 lset
- Files share pages through global buffer table

Parameters

- N – total number of buffers
- l_{ij} – max number of buffers for file instance j of query i (desired size)
- r_{ij} –number of buffers allocated for file instance j of query i (actual size)

DBMIN Algorithm

Query requests page => search global table:

1) Found in global table and locality set
   - Update usage stats
2) In memory, not in locality set
   - If already owned **by someone else**, return it
   - Else, return to locality set and increment r_{ij}
   - If $r_{ij} > l_{ij}$, release a page to global free list

DBMIN Algorithm (cont.)

3) Not in memory
- Get a free buffer
- Schedule a read, then do "in memory" (step 2)

On file open/close, do load control:

(Open): if $\Sigma_i \Sigma_j l_{ij} < N$, query can proceed, otherwise waits

(Close): release buffers to free list, unblock one or more waiters

Chou and DeWitt - Outline

- Review of Algorithms
  - Domain separation (Reiter)
  - ...
- DBMIN - ideas and **algorithms**
- Experiments

Performance Results

Compared to
- Rand
- FIFO
- Clock
- WS
- Hot Set
- DBMIN

Workload

Queries:

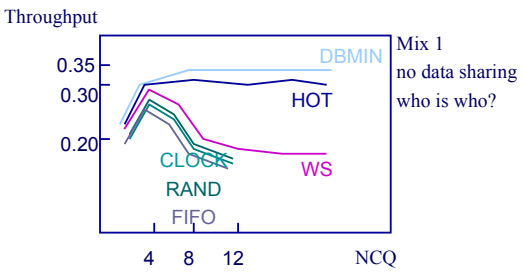
1) q1: selection (clustered index)
2) q2: selection (non-clustered index)
3) q3: selection (cl-index) + join (index-join)
4) q4: seq scan + index join (non-cl-index)
5) q5: selection (cl-index) + join (nested loops)
6) q6: selection (cl-ind) + hash join

Workload

Mixes:

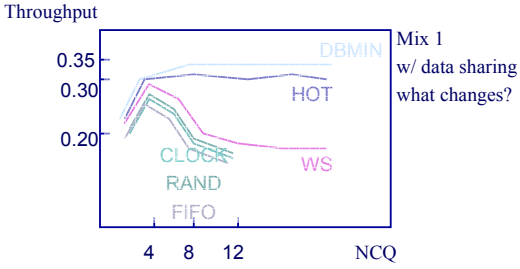
1) Mix1: all 6 queries equally
2) Mix2: more of q1 and q2 (selections)
3) Mix3: much more of q1, q2

Typical results



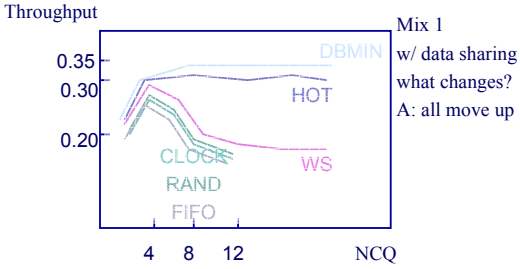
Mix 1
no data sharing
who is who?

Typical results



Mix 1
w/ data sharing
what changes?

Typical results



Mix 1
w/ data sharing
what changes?
A: all move up

Typical results

- What about the 'lighter', M2 and M3 mixes?

Typical results

- What about the 'lighter', M2 and M3 mixes?
- A: similar performance (higher throughput)

Performance Results (cont.)

- DBMIN did best
- Hot set was next
- WS was next (trouble with join loops)
- Then: clock, FIFO, rand (thrashing as multiprogramming level increases)
- Load control helps
- However: too complex
- Believe it or not, better algorithms just appeared (2Q, ARC)!