

# An Evaluation of Buffer Management Strategies for Relational Database Systems

Chou, DeWitt

Presenter: Anand Sivaram

10/30/09

EECS 584, Fall 2009

1

## Outline

- Motivation
- Previous Buffer Management Models
- The Query Locality Set Model
- The DBMin Algorithm
- Evaluation
- Conclusion

10/30/09

EECS 584, Fall 2009

2

## Motivation

- Goal: Keep “useful” things in memory, decide what to discard.
- Observation: Databases use combinations of regular operations with predictable reference patterns.
- OS doesn’t know anything about reference patterns of query, so it’s policies will be too generic.
- Need buffer management policies that “understand” database systems - exploit predictability of reference behavior by looking at structure of queries.
- DBMin: Classify reference patterns, use these to decide how much space buffer space is needed, and what replacement policy to use.

10/30/09

EECS 584, Fall 2009

3

## Previous Models: Domain Separation

- Idea: Some types of objects more important than others. Example: Root of B+ tree index during indexed access.
- Model: Pages classified into types, each managed separately. Example: One domain for each non-leaf level of the B+ tree, and one domain for all leaf pages.
- Challenges:
  - Domains are static (not adaptable to requirements of queries) [eg data page repeatedly accessed - vs - accessed once.]
  - Does not differentiate between relative importance of types.
  - Interference among different users
  - No load control.

10/30/09

EECS 584, Fall 2009

4

## “New” Algorithm

- Ideas:
  - Priority of a page is a property of the relation.
  - Each relation needs a working set.
- Model:
  - Each active relation assigned a resident set – initially empty.
  - Resident sets linked in priority queue, with global free list on top (least priority).
  - MRU used within each set.
- Challenges:
  - How do we decide priority?
  - Searching through priority list can be expensive.
  - MRU not always good.
  - How do we determine priority in multi user environments?

10/30/09

EECS 584, Fall 2009

5

## Hot Set Algorithm

- Idea: Identify set of pages over which there is looping behavior (“Hot Set”) – try to keep these pages in memory.
- Model:
  - Calculate hot set for each query. Allocate that much buffer space to it. Example: nested-loop join  $R \bowtie S$ , hot set is number of pages in  $S + 1$
  - New queries wait till its |Hot Set| space becomes available.
  - Use LRU replacement within each partition;
- Challenges:
  - Based on the assumption of using LRU for replacement. LRU is the worst choice in some cases (especially for looping behavior)
  - Tends to over-allocate memory.

10/30/09

EECS 584, Fall 2009

6

## DBMin: Query Locality Set Model

- Databases use limited operations, each with a particular reference pattern. Basic operations can be combined to form all other operations.
- So reference behavior of database operations can be described as composition of a set of **simple and regular reference patterns**.
- Number of pages to be kept in memory ("**Locality Set Size**") depends on the reference pattern.
- Most appropriate **page replacement policy** also depends on the reference pattern.

10/30/09

EECS 584, Fall 2009

7

## Query Locality Set Model: Reference Patterns

- **Straight Sequential**: Pages referenced by sequential access of a relation.
- Example: Select on an unordered relation.
- Locality Set Size: Single page frame (since the same page is never read again)
- Replacement Policy: Replace each page each time.

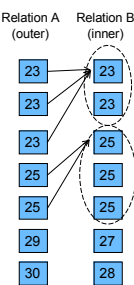
10/30/09

EECS 584, Fall 2009

8

## Reference Patterns

- **Clustered Sequential**: Some sequential regions of a relation scanned repeatedly.
- Example: In a merge join, if both relations have several records with same key, the record set with same value in inner relation is repeatedly scanned for each of the equal values in the outer relation.
- Locality Set size:  
|Records in largest cluster| /  
Blocking Factor
- Replacement Policy: FIFO / LRU



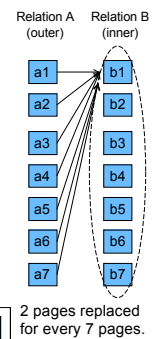
10/30/09

EECS 584, Fall 2009

9

## Reference Patterns

- **Looping Sequential**: Repeated reference to whole file.
- Example: Inner relation in nested loop join.
- Locality Set Size: Number of pages in inner relation (or as many pages as possible)
- Replacement Policy: MRU (since the most recently used page will be read again only after all other pages have been read once)



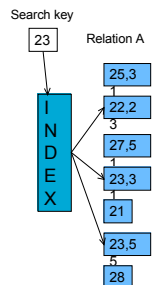
10/30/09

EECS 584, Fall 2009

10

## Reference Patterns

- **Independent Random**: Series of independent accesses on a relation.
- Example: Indexed scan (non-clustered)
- Locality Set size:
  - Can be calculated using Yao's formula (gives approximate upper bound on number of pages referenced in a series of random references)
- Replacement: Any policy



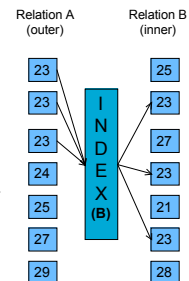
10/30/09

EECS 584, Fall 2009

11

## Reference Patterns

- **Clustered Random**: Repeated access to random pages.
- Example: Indexed Nested Loop Join with non-clustered, non-unique index on inner relation, clustered, non-unique index on outer.
- Locality Set Size: Number of records in largest cluster (can be calculated using Yao's formula)
- Replacement Policy: Any Policy



10/30/09

EECS 584, Fall 2009

12

## Reference Patterns

- **Straight Hierarchical:** Traversing root to leaves of an index while retrieving a single tuple.
- Locality Set Size: 1 (since traversed only once)
- Replacement Policy: Replace each page.
- **Hierarchical with straight sequential:** Index traversal followed by straight sequential on leaves
- Locality Set Size: 1
- Replacement Policy: Replace each page
- **Hierarchical with clustered sequential:** Index traversal followed by clustered sequential on leaves
- Locality Set Size: number of index pages in the largest cluster
- Replacement Policy: FIFO or LRU

10/30/09

EECS 584, Fall 2009

13

## Reference Patterns

- **Looping Hierarchical:** Repeated traversal of Index
- Example: Repeated index scan during nested loop join with indexed inner relation
- Locality Set size: Use Yao's formula to decide how many levels of index to keep in memory.
- Pages closer to root more important. (In most cases, keeping just root is most feasible)
- LIFO with 3-4 buffers appropriate in most cases.

10/30/09

EECS 584, Fall 2009

14

## DBMin

- Buffers allocated per file instance. File instance owns a set of buffers.
- Set of pages owned by a file instance = Locality Set
- Size of locality set determined by reference pattern.
- Reference behavior model not tied to any one page replacement strategy. Most appropriate strategy chosen based on reference pattern.
- For data sharing – each buffer also accessible through global buffer table.

10/30/09

EECS 584, Fall 2009

15

## DBMin (Cont'd)

- $N$  – Total number of buffers in the system.
- $l_{ij}$  – max buffers that can be allocated to file instance  $j$  of query  $i$  (*desired locality set size*)
- $r_{ij}$  – buffers currently allocated to file instance  $j$  of query  $i$
- Estimate locality set sizes by examining the query plan and database statistics.
- Activate load controller when any file opened/closed. Check:  $\sum \sum l_{ij} < N$  for all active queries  $i$ , file instances  $j$ . (Basically, see if |locality sets| available.) If yes, allow query to proceed, otherwise, make it wait.

10/30/09

EECS 584, Fall 2009

16

## Handling a Page Request

1. Found in Global and Local Sets: Update statistics (if and as required by the local page replacement policy)
2. Found in memory, but not in locality set:
  - A) In some other locality set: just use the page
  - B) In global list:
    - Add to locality set.  $r++$ .
    - If  $(r>1)$  release a page to global free list according to local replacement policy). Set  $r = 1$ .
    - Update statistics.
3. Not found in Memory: Read from disk into global free list. Then do as in step 2.

10/30/09

EECS 584, Fall 2009

17

## Evaluation

- Simulation used for evaluating the algorithm
- Hybrid Simulation Model:
  - Trace-driven simulation: Record trace strings for queries from a real system, tracking the statistics on the following:
    - Page Read / Write
    - Disk Read / Write
    - File Open, File Close
  - Distribution-driven simulation
    - Events generated by a stochastic model
- System workload synthesized by merging trace strings of concurrent queries.

10/30/09

EECS 584, Fall 2009

18

## Evaluation

- Configuration Model – Database System Simulator
  - Manages resources:
    - CPU
    - I/O Device
    - Memory
- Performance Measurement – Throughput (Average number of queries completed per second)

10/30/09

EECS 584, Fall 2009

19

## Evaluation

- Three factors affecting throughput in multiuser environment:
  - Number of concurrent queries (NCQ) – varied from 1 to 32
  - Degree of data sharing
    - 32 copies of database replicated on the disk
      - Full sharing – all queries access the same database
      - Half sharing – every 2 queries share a copy of the database
      - No sharing – every query has its own copy of the database
  - Query mix
    - CPU Demand
    - Disk Demand
    - Memory Demand

10/30/09

EECS 584, Fall 2009

20

## Query Mix

Query Type	CPU Demand	Disk Demand	Memory Demand
I	Low	Low	Low
II	Low	High	Low
III	High	Low	Low
IV	High	High	Low
V	High	Low	High
VI	High	High	High

- M1: All 6 equally likely
- M2: I and II chosen 50% of the time
- M3: I and II chosen 75% of the time

Query #	Query Operators	Selectivity	Access Path of Selection	Join Method	Access Path of Join
I	select(A)	1%	clustered index	-	-
II	select(B)	1%	non-clustered index	-	-
III	select(A) join B	2%	clustered index	index join	clustered index on B
IV	select(A') join B	10%	sequential scan	index join	non-clustered index on B
V	select(A) join B'	3%	clustered index	nested loops	sequential scan over B'
VI	select(A) join A'	4%	clustered index	hash join	hash on result of select(A)

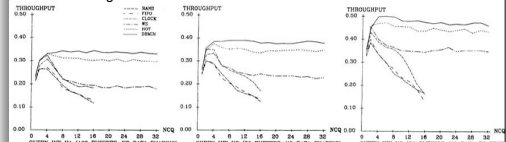
10/30/09

EECS 584, Fall 2009

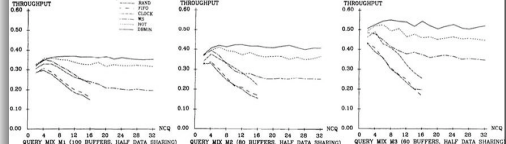
21

## Comparison with other Algorithms

### No Data Sharing



### Half Data Sharing



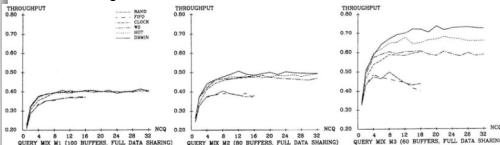
10/30/09

EECS 584, Fall 2009

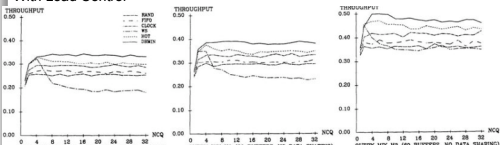
22

## Comparison with other Algorithms

### Full Data Sharing



### With Load Control



10/30/09

EECS 584, Fall 2009

23

## Some Observations

- DBMin's throughput always highest, followed by Hot Set.
  - Attributed to predictive nature of these algorithms (using knowledge of reference query patterns)
- Data sharing improved performance of all algorithms
- Load Control also helped improve performance of the simpler algorithms.

10/30/09

EECS 584, Fall 2009

24



## Conclusion

- Some basic reference patterns used a lot in relational databases. All operations are combinations of these patterns.
- Pattern deciphered from query structure and relations accessed.
- Buffer space required, as well as page replacement policy, are functions of these access patterns.
- Use knowledge of these patterns to manage buffer more intelligently.