CS 764: Topics in Database Management Systems
Lecture 3: Buffer Management

Xiangyao Yu
9/14/2020
Is it possible to make GRACE hash join work when $|M| < \sqrt{|R| \times F}$? For example, $|M| = 10$, $F = 1$, $|R| = 1000$. You may modify the GRACE hash join algorithm as described in the paper.

- Multiple phases of partitioning. For $k$ partition phases, we can get $|M|^k$ partitions

Is it possible for a sort-merge join algorithm to outperform a hash-based join algorithm? If yes, when can this happen?

- Sort-merge join can out-perform hash-based join when both relations are already sorted based on the join key
An Evaluation of Buffer Management Strategies for Relational Database Systems

Hong-Tai Chou
David J. DeWitt

Computer Sciences Department
University of Wisconsin

ABSTRACT
In this paper, we present a new algorithm, DBMIN, for managing the buffer pool of a relational database management system. DBMIN is based on a new model of relational query behavior, the query locality set model (QLSM). Unlike the hot set model, the QLSM has an advantage over the stochastic models due to its ability to predict future reference behavior. However, the QLSM avoids the potential problems of the hot set model by separating the modeling of reference behavior from any particular buffer management algorithm. After introducing the QLSM and describing the DBMIN algorithm, we present a performance evaluation methodology for evaluating buffer management algorithms in a multiuser environment. This methodology employed a hybrid model that combines features of both trace driven and distribution driven simulation models. Using this model, the performance of the DBMIN algorithm in a multiuser environment is compared with that of the hot set algorithm and four new model of relational query behavior, the query locality set model (QLSM). Like the hot set model [Sace82], the QLSM has an advantage over the stochastic models due to its ability to predict future reference behavior. However, the QLSM avoids the potential problems of the hot set model by separating the modeling of reference behavior from any particular buffer management algorithm. After introducing the QLSM and describing the DBMIN algorithm, the performance of the DBMIN algorithm in a multiuser environment is compared with that of the hot set algorithm and four more traditional buffer replacement algorithms.

A number of factors motivated this research. First, although Stonebraker [Ston81] convincingly argued that conventional virtual memory page replacement algorithms (e.g., LRU) were generally not suitable for a relational database environment, the area of buffer management has, for the most part, been ignored (contrast the activity in this area with that in the concurrency control area). Second, while the hot set...
Agenda

Buffer management basics

Query locality set model (QLSM)

DBMIN algorithm

Other buffer management algorithms

Evaluation
Buffer Management Basics
Basic Concepts (covered in CS 564)

A database management system (DBMS) manipulate data in memory
- Data on disk must be loaded to memory before processed

The unit of data movement is a page

Page replacement policy (what pages should stay in memory?)
- LRU (Least recently used)
- Clock
- MRU (Most recently used)
- FIFO, Random, …
LRU Replacement Example

Example: memory contains 4 buffers. LRU replacement policy

Memory

Incoming requests
0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, …
LRU Replacement Example

Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, …</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td>Page 3</td>
<td></td>
</tr>
</tbody>
</table>

Cold start misses: load pages 0—3 to memory
LRU Replacement Example

Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, …</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td>Page 3</td>
<td></td>
</tr>
</tbody>
</table>

Cache hits on pages 0—2
Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Page 0</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Page 3</td>
</tr>
</tbody>
</table>

Incoming requests:

0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, ...

Page 4 replaces page 3 in the buffer since page 3 is the least-recently used page
LRU Replacement Example

Example: memory contains 4 buffers. LRU replacement policy

Memory

| Page 0 | Page 1 | Page 2 | Page 4 |

Incoming requests

0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, ...

Cache hits on pages 0—2
Example: memory contains 4 buffers. LRU replacement policy

Example: memory contains 4 buffers. LRU replacement policy

In memory, there are 4 buffers:

- Page 0
- Page 1
- Page 2
- Page 4
- Page 5

Incoming requests:

0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, ...

Page 5 replaces page 4 in the buffer since page 4 is the least-recently used page.
A Different Access Pattern

Example: memory contains 4 buffers. LRU replacement policy

Memory

Incoming requests

0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ...

Disk
A Different Access Pattern

Example: memory contains 4 buffers. LRU replacement policy

| Page 0 | Page 1 | Page 2 | Page 3 |

Incoming requests

0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ...

Cold start misses: load pages
0—3 to memory
A Different Access Pattern

Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 4</td>
<td>0, 1, 2, 3, 4, 0, 1, 2, 3, 4, …</td>
</tr>
<tr>
<td>Page 1</td>
<td>Page 2</td>
</tr>
<tr>
<td>Page 3</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>Page 4 replaces page 0 since page 0 is the least-recently used page</td>
</tr>
</tbody>
</table>
A Different Access Pattern

Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 4</td>
<td>0, 1, 2, 3, 4, 0, 1, 2, 3, 4, …</td>
</tr>
<tr>
<td>Page 0</td>
<td>Page 2</td>
</tr>
<tr>
<td>Page 3</td>
<td>Page 1</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
</tbody>
</table>

Page 0 replaces page 1 since page 1 is the **least-recently used** page

Each future access will replace the page that will be immediately accessed
A Different Access Pattern

Example: memory contains 4 buffers. LRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 4</td>
<td>Page 0</td>
</tr>
<tr>
<td>0, 1, 2, 3, 4, 0, 1, 2, 3, 4, …</td>
<td></td>
</tr>
</tbody>
</table>

Page 0 replaces page 1 since page 1 is the least-recently used page.

Each future access will replace the page that will be immediately accessed.

Under LRU, all accesses in this pattern are cache misses!
MRU Replacement Example

Example: memory contains 4 buffers. MRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>Page 1</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>Page 2</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>Page 3</td>
<td>0, 1, 2, 3, 4</td>
</tr>
</tbody>
</table>

Disk
MRU Replacement Example

Example: memory contains 4 buffers. MRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0, 1, 2, 3, 4, 0, 1, 2, 3, 4, …</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td>Page 4</td>
<td></td>
</tr>
<tr>
<td>Page 3</td>
<td></td>
</tr>
</tbody>
</table>

Disk

Page 4 replaces page 3 since page 3 is the most-recently used page
MRU Replacement Example

Example: memory contains 4 buffers. MRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
<th>Incoming requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ...</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td>Page 4</td>
<td></td>
</tr>
</tbody>
</table>

Cache hits on pages 0—2
**MRU Replacement Example**

Example: memory contains 4 buffers. MRU replacement policy

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
</tr>
</tbody>
</table>

Page 3 replaces page 2 since page 2 is the most-recently used page

Incoming requests:

0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ...

LRU: all accesses are misses
MRU: 25% of accesses are misses

Selection of replacement policy depends on the data access pattern
Query Locality Set Model (QLSM)
Query Locality Set Model

Observations

• DBMS supports a limited set of operations
• Data reference patterns are regular and predictable (e.g., from parser)
• Complex reference patterns can be decomposed into simple patterns
Query Locality Set Model

Observations
- DBMS supports a limited set of operations
- Data reference patterns are regular and predictable
- Complex reference patterns can be decomposed into simple patterns

Reference pattern classification
- Sequential
- Random
- Hierarchical

Locality set: the appropriate buffer pool size for each query
Straight sequential (SS): each page in a file accessed only once
  • E.g., select on an unordered relation
  • Locality set: one page
  • Replacement policy: any
QLSM – Sequential References

Straight sequential (SS): each page in a file accessed only once
  • E.g., select on an unordered relation
  • Locality set: one page
  • Replacement policy: any

Clustered sequential (CS): repeatedly read a “chunk” sequentially
  • E.g., sort-merge join with duplicate join keys
  • Locality set: size of largest cluster
  • Replacement policy: LRU or FIFO (buffer size $\geq$ cluster size), MRU (otherwise)
Straight sequential (SS): each page in a file accessed only once
  • E.g., select on an unordered relation
  • Locality set: one page
  • Replacement policy: any

Clustered sequential (CS): repeatedly read a “chunk” sequentially
  • E.g., sort-merge join with duplicate join keys
  • Locality set: size of largest cluster
  • Replacement policy: LRU or FIFO (buffer size ≥ cluster size), MRU (otherwise)

Looping Sequential (LS): repeatedly read something sequentially
  • E.g. nested-loop join
  • Locality set: size of the file being repeated scanned.
  • Replacement policy: MRU
QLSM – Random References

Independent random (IR): truly random accesses

- E.g., index scan through a non-clustered (e.g., secondary) index
- Locality set: one page or $b$ pages ($b$ unique pages are accessed in total)
- Replacement: any
QLSM – Random References

Independent random (IR): truly random accesses
• E.g., index scan through a non-clustered (e.g., secondary) index
• Locality set: one page or $b$ pages ($b$ unique pages are accessed in total)
• Replacement: any

Clustered random (CR): random accesses with some locality
• E.g., join between non-clustered, non-unique index as inner relation and clustered, non-unique outer relation
• Locality set: size of the largest cluster
• Replacement policy:
  LRU or FIFO (buffer size $\geq$ cluster size)
  MRU (otherwise)
Straight hierarchical (SH): single traversal of the index  
  • Similar to SS

Hierarchical with straight sequential (H/SS): traversal followed by straight sequential on leaves  
  • Similar to SS

Hierarchical with clustered sequential (H/CS): traversal followed by clustered sequential on leaves  
  • Similar to CS

Looping hierarchical (LH): repeatedly traverse an index  
  • Example: index nested-loop join  
  • Locality set: first few layers in the B-tree  
  • Replacement: LIFO
## Summary of Reference Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
<th>Locality set</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight sequential (SS)</td>
<td>File scan</td>
<td>1 page</td>
<td>any</td>
</tr>
<tr>
<td>Clustered sequential (CS)</td>
<td>Sort-merge join with duplicate keys</td>
<td>Cluster size</td>
<td>LRU/FIFO</td>
</tr>
<tr>
<td>Looped sequential (LS)</td>
<td>Nested-loop join</td>
<td>Size of scanned file</td>
<td>LRU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; Size of scanned file</td>
<td>MRU</td>
</tr>
<tr>
<td>Independent random (IR)</td>
<td>non-clustered index scan</td>
<td>1 or b</td>
<td>any</td>
</tr>
<tr>
<td>Clustered random (CR)</td>
<td>Non-clustered, non-unique index as inner relation in a join</td>
<td>Same as CS</td>
<td></td>
</tr>
<tr>
<td>Straight hierarchical (SH)</td>
<td>Single index lookup</td>
<td></td>
<td>Same as SS</td>
</tr>
<tr>
<td>Hierarchical with straight sequential (H/SS)</td>
<td>Index lookup + scan</td>
<td></td>
<td>Same as SS</td>
</tr>
<tr>
<td>Hierarchical with clustered sequential (H/CS)</td>
<td>Index lookup + clustered scan</td>
<td></td>
<td>Same as CS</td>
</tr>
<tr>
<td>Looping hierarchical (LH)</td>
<td>Index nested-loop join</td>
<td>First few layers in the B-tree</td>
<td>LIFO</td>
</tr>
</tbody>
</table>
DBMIN algorithm
For each open file operation
  • Allocate a set of buffers (i.e., locality set)
  • Choose a replacement policy
  • Each open file instance has its own set of buffers
  • If two file instances access the same page, they share the page

Predicatively estimate locality set size by examining the query plan and database statistics

Admission control: a query is allowed to run if its locality sets fit in free frames
Other Buffer Management Algorithms
Simple Algorithms

Replacement discipline is applied globally to all the buffers in the system

• RAND
• FIFO (first-in, first-out)
• CLOCK
Sophisticated Algorithms

Replacement discipline is applied locally to each query or file instance

- DBMIN
- HOT (the hot set algorithm): always using LRU
- WS (the working set algorithm)
- Domain separation: LRU within each domain (e.g., an index level)
Evaluation

Except DBMIN and HOT, performance of all the other algorithms thrashes at high concurrency.

DBMIN outperforms HOT.
Q/A – Buffer Management

Complexity of DBMIN over other algorithms?

What is a file instance?

Two file instances access the same page?
  • Eviction due to owner but the other query still relies on it

Memory pages vs. global free list vs. locality set?

Modern RDBMSs use simple replacement policy?

Can we use an ML model instead to predict the reference patterns?

Optimize across file instances?

What is thrashing?
Group Discussion

Consider a nested loop join between R and S. Initially R and S are both stored on disk. The buffer management policy is DBMIN.

- $|R| = 4$
- $|S| = 10$
- $|M| = 6$

Q1: How many pages need to be read from disk to perform the join?

Q2: Does the answer to Q1 change when $|M| = 4$? What is the buffer management policy for R and S in this case?
Before Next Lecture

Submit discussion summary to https://wisc-cs764-f20.hotcrp.com

- Title: Lecture 3 discussion. group ##
- Authors: Names of students who joined the discussion
- Summary submission Deadline: Tuesday 11:59pm

Before next lecture, submit review for