Weaving Relations for Cache Performance

Anastassia Ailamaki
David J. DeWitt
Mark D. Hill
Marios Skounakis

Presented by:
Ippokratis Pandis
Bottleneck in DBMSs

- Processor speeds increase faster than memory speeds
- DBMSs becoming compute and memory bound
  - New architectures hide I/O latency
  - Larger memories
  - Pipelining/Out of order execution/Non-blocking caches
  - Complexity of DBMSs
- Major performance bottleneck are the L2 data cache misses
Data cache misses

- Requests for data not found in the cache and fetched from the main memory
- Major penalty in comparison to L1 cache misses
- Loading cache with useless data:
  - Wastes bandwidth
  - Pollutes the cache
  - Possibly forces replacement of useful information
  - Increases the number of cache misses
Data placement policies

- **NSM (N-ary Storage Model)**
  - Stores records sequentially within a page
  (+) Intuitive and Easy to implement
  (-) Exhibits poor cache performance

- **DSM (Decomposition Storage Model)**
  - A n-attribute relation is partitioned into n subrelations
  (+) Exploits spatial locality
  (-) Huge reconstruction cost (joining participating subrelations together)
New Data Placement Policy

- PAX stands for Partition Attributes Across
- The records are partitioned within each page storing together values of each attribute in minipages
- This design:
  - Exhibits fewer L2 data cache misses
  - Executes faster range selection queries and updates
  - Executes faster TPC-H queries involving I/Os
  - Disregarding the number of participating attributes
Outline

- Motivation
- Related Work
- PAX
- Experimental results
- Summary
- Discussion
NSM [1] – Intro

- NSM – N-ary Storage Model
- Records are stored sequentially
- Pointers (offset) to end of page
- One offset of each variable length entry
- Intra-record locality (attributes of record r together)
- Intuitive
- Simple to implement
- Pollutes cache memory
NSM [2] – Record Design

- All attributes of a record are stored together
- NSM pollutes the cache and wastes bandwidth
NSM [3] – Cache Behavior

Given the SQL query:

```
select name from R
where age > 40
```

The database has a main memory and a cache. The cache contains:

- block 1: Jane 30 RH
- block 2: RH3 1563
- block 3: Jim 20 RH4
- block 4: 52 2534 Leon

The main memory contains:

- Jane 30 RH
- 45 RH3 1563
- 7658 Susan 52 2534 Leon

The query will access the cache for Jane and John, as they are in the cache.
DSM [1] – Intro

- DSM – Decomposition Storage Model
- Store n-attribute table as n-one attribute tables
- High cache utilization
- But, High reconstruction cost

MAIN MEMORY

PAGE HEADER

1  Jane  2
John  3  Jim  4  Suzan

block 1
1  30  2  45

block 2
3  20  4  52  5

CACHE

15-823 Hot Topics in DB Systems
Outline

• Motivation
• Related Work
• PAX
• Experimental results
• Summary
• Discussion
PAX [1] – Intro

- PAX – Partition Attributes Across
- Partition data within the page for spatial locality
- Fewer cache misses, low reconstruction cost
- Implementing PAX requires changes only to the page-level data manipulation code
- Can be used interchangeable with other data layout schemes
PAX [2] – Comparison with NSM

NSM

<table>
<thead>
<tr>
<th>PAGE HEADER</th>
<th>RH1</th>
<th>1237</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>30</td>
<td>RH2</td>
</tr>
<tr>
<td>45</td>
<td>RH3</td>
<td></td>
</tr>
<tr>
<td>7658</td>
<td>Susan</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>RH3</td>
<td></td>
</tr>
<tr>
<td>1563</td>
<td>Jim</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RH4</td>
<td></td>
</tr>
</tbody>
</table>

PAX

<table>
<thead>
<tr>
<th>PAGE HEADER</th>
<th>1237</th>
</tr>
</thead>
<tbody>
<tr>
<td>1563</td>
<td></td>
</tr>
<tr>
<td>7658</td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td></td>
</tr>
<tr>
<td>Susan</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

15-823 Hot Topics in DB Systems

select name from R
where age > 40
### PAX [4] – Detailed View

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Sizes</th>
<th>Free Space</th>
</tr>
</thead>
<tbody>
<tr>
<td># records</td>
<td># attributes</td>
<td>attribute sizes</td>
</tr>
<tr>
<td>presence bits</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>pid</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>v</th>
<th>4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1237</td>
<td>4322</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td>John</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 30 | 45 |```
Outline

- Motivation
- Related Work
- PAX
- Experimental results
- Summary
- Discussion
Experimental Setup

- Windows NT4.0
- Pentium II Xeon/MT Workstation
- 16KB L1-I, 16KB L1-D, 512 KB L2, 512 MB RAM
- Minimal setup
  - Memory resident DB
  - Exclusion of I/O subsystem
- Processor hardware counters for the experiments
- Implemented on top of Shore Storage Manager
NSM/DSM/PAX Comparison

- Elapsed time while number of attributes or selectivity in query increases
- DSM becomes unaffordable
Execution Time Breakdown

- PAX in comparison with NSM exhibits less:
  - Data cache misses
  - L2 Data penalty
  - L1 Data penalty
Sensitivity Analysis

- Sensitivity to projectivity
- Sensitivity to number of attributes in relation
- As the number of attributes involved in the query increases the performance of the two techniques converges
Evaluation with DSS Workloads

- Windows NT4.0
- Pentium II Xeon/MT Workstation
- 16KB L1-I, 16KB L1-D, 512 KB L2, 512 MB RAM
- Processor hardware counters for the experiments
- Implemented on top of Shore Storage Manager
- Loaded 100M, 200M, and 500M TPC-H DBs
- Ran Queries:
  - Range Selections with variable parameters (RS)
  - TPC-H Q1 and Q6
    - sequential scans
    - lots of aggregates (\textit{sum}, \textit{avg}, \textit{count})
    - grouping/ordering of results
  - TPC-H Q12 and Q14
    - (Adaptive Hybrid) Hash Join
    - complex ‘where’ clause, conditional aggregates
Bulk-loading/Insertions

- Estimate average field sizes
- Start inserting records
- If a record doesn’t fit,
  - Reorganize page
  - (move minipage boundaries)
- Adjust average field sizes

- 50% of reorganizations accommodate a single record (the last record in the page)

- PAX loads a TPC-H database in 2-26% longer than NSM
PAX/NSM Speedup

PAX improves performance up to 42% even with I/O
Outline

- Motivation
- Related Work
- PAX
- Experimental results
- Summary
- Discussion
Summary [1]

- DBMSs do not exploit architecture features of modern processors
- PAX offers a new technique for the placement of data
- PAX exhibits significant improvement in the number of L2 data cache misses
- Number of comparisons with NSM and DSM present better performance:
  - High data cache performance
  - Faster than NSM for DSS queries
- Orthogonal to other storage decisions
- Does not affect I/O performance
## Summary [2] – Comparison

<table>
<thead>
<tr>
<th>Criterion</th>
<th>NSM</th>
<th>DSM</th>
<th>PAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-record spatial locality</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Low record reconstruction cost</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Implementation</td>
<td>SIMPLE</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Performance with large number of attr.</td>
<td>GOOD</td>
<td>BAD</td>
<td>GOOD</td>
</tr>
</tbody>
</table>

*Additions?*
Outline

- Motivation
- Related Work
- PAX
- Experimental results
- Summary
- Discussion
Technical Questions

- Why there is no space overhead in comparison with NSM?
- Does the underlying platform affect the performance? (Linux/AMD)

Other Questions?
Critique

- Difficult Implementation
- No results regarding TPC-C (OLTP) workload presented. – Any data layout policy should exhibit relatively good performance in both the two types of workloads
- What happens after deletions? Overhead for page re-organization
- What is the behavior with the variable-length attributes? Recalculation of boundaries.
- Experimental setup very convenient for PAX. Fixed-line queried attributes, exactly 4 times smaller than cache block size.
Points of Discussion

- Has this idea been implemented in any commercial product?
- Work for addressing the L1I cache misses problem? Or it is only task of the compiler?
- How about the new trends in computer architecture?
Thank you!!

ipandis@cs.cmu.edu