Data Placement in the FATES Automatically-Tuned Database Storage Manager

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

Current DB Storage Managers

- multi-level storage hierarchy
- different devices at each level
- different ways to access data on each device
- variable workloads and access patterns
- device and workload-specific data placement

Goal: Reduce data transfer costs in memory hierarchy
Maximize utilization in memory and in processor cache

Planning and Executing Queries

- Optimizer
  - query plan

- Execution
  - query at a time
  - many queries at a time

Goal: Restore optimizer's assumptions
Minimize manual tuning
Database storage managers should...
- minimize memory and disk delays
- maximize utilization at all storage levels
- restore query planner’s assumptions
- export useful device characteristics
- encapsulate functionality for flexibility
- minimize manual tuning

The Fates DB Storage Manager

Efficient execution at all levels of memory hierarchy

Outline
- Introduction
- Data Layout in DBMS
  - Clotho Storage Model
  - Atropos Logical Volume Manager
- Experimental Results
- Conclusions
Previous Page Layouts
- N-ary Storage Model (NSM)
  - Records stored sequentially in pages
  - Optimized for full-record access
- Decomposition Storage Model (DSM)
  - Each attribute stored as a separate relation
  - Good for only partial-record access
- Partition Attributes Across (PAX)
  - In-page DSM
  - Good CPU cache performance
  - Inherits I/O performance from NSM

Each optimizes for a subset of applications/devices

“Classic” Disk Page Layout
- NSM (n-ary Storage Model, or Slotted Pages)
  - Records are stored sequentially
  - Attributes of a record are stored together

Records are stored sequentially
Attributes of a record are stored together

NSM in Memory Hierarchy
- Optimized for full-record access
- Slow partial-record access
- Wastes I/O bandwidth (fixed page layout)
- Wastes memory bandwidth (garbage in cache)
Partition original table into $n$ 1-attribute sub-tables

Each sub-table stored separately in NSM pages

DSM (Decomposition Storage Model)

DSM in Memory Hierarchy

- Query accesses all attributes (full-record access)
- Slow full-record access
- Reconstructing full record may incur random I/O
- Optimized for partial-record access

Costly

- $\beta$

$\begin{align*}
\text{select} & \quad \text{name} \\
\text{from} & \quad \text{R} \\
\text{where} & \quad \text{age} > 50
\end{align*}$
Idea: Partition data within page for spatial locality.

PAX in Memory Hierarchy

- Partial-record access in memory
- Full-record access on disk

- Optimizes cache-to-memory communication
- Eliminates unnecessary trips to memory
- Retains NSM’s I/O (page contents do not change)

Summary (no replication)

Need new placement method:
- Efficient full- and partial-record accesses
- Maximize utilization at all levels of memory hierarchy

Difficult!!! Different devices/access methods

Different workloads on the same database
Outline

- Introduction
- Data Layout in DBMS
- Clotho Storage Model
  - CSM page layout
  - buffer pool management
  - Atropos Logical Volume Manager
  - Experimental Results
- Conclusions

Clotho Storage Model (CSM) [SSS04]

**CSM: In-Memory Page**

- SELECT * From R Page with all attributes
- SELECT Name From R Page with attr. Name
- SELECT EID From R WHERE Age > 52 Page with attr. EID & Age

In-memory page contents tailored to query

**CSM: On-disk (assumed) page** [SSS04]

- PAX Page on Disk
- CSM Page on Disk

CSM on disk: Similar to PAX layout
Minipages aligned to block boundary
Clotho: CSM page composition

- Use (page_id, p_sch) to look up records
- Hit: page_id matches and (p_sch ⊆ schema)
- Page schema management for queries
  - When new query comes
    - If (exist p_sch ⊆ q_sch) use existing p_sch
    - else if (q_sch disjoint with all p_sch) add q_sch to p_sch list
    - else new p_sch = union(q_sch, all p_sch overlapping with q_sch)
  - When query finishes
    - Remove q_sch, adjust p_sch list
- Write statements (update/insert/delete)
  - Use full schema
  - Invalidate frames w/ same page_id, any schema

In-memory “skeleton” tailored to query
- Just the data you need
- Query-specific pages
- Great cache performance
- Decoupled layout
- Fills different hardware
- Low reconstruction cost
- Done at I/O level
- Guaranteed by Lachesis and Atropos [SSS04a]

On-disk page:
- PAX-like layout
- Block boundary aligned

Sharing and consistency in buffer pool?

Clotho: Buffer Pool Management

- Time to look at the disk...

Need device-aware data placement!!!

Table R

LBNs on disk (512 B/blk)

Maps to LBNs

Efficient partial-record access (sequential access)

Minimal bookkeeping / maximum sharing
Outline

- Introduction
- Data Layout in DBMS
- Clotho Storage Model
- Atropos Logical Volume Manager
  - basic idea
  - logical organization
  - Efficient disk accesses with Lachesis
- Experimental Results
- Conclusions

Atropos Logical Volume Manager

- Provides logical-to-physical mapping
- Issues I/Os to individual disks
- Exposes important device attributes
- Exploits device-specific disk characteristics
  - Track-aligned access
  - Semi-sequential access
- Accesses 2D data structures efficiently

Storing 2D data structures

Linear representation

Track length = 4
We do not have to use the naïve layout!

Because we decouple the memory layout from storage organization, we can choose a better on-disk layout and avoid random access along the secondary axis.
Semi-sequential access
- Exploits knowledge of head switch time
- Also proposed in [Gorbatenko2002]

Each access is followed by a disk head switch.

Mapping semi-sequential LBNs
- Four tracks – 21 sectors per track
- Single quadrangle – b = 2, w = 10, d = 4

On next track access head reads next attribute.

Mapping to physical disk locations
- Four tracks – 21 sectors per track
- Track skew – H = 3 disk blocks

Given b, H, and N, d = 4
Issuing requests

- Issue d requests to the disk in a batch
- Scheduler chooses initial request
- Initial seek time, rotational latency
- Remaining requests serviced semi-sequentially

Access efficiency for 2D data

- More than one revolution
- Semi-sequential access much better than random

Atropos on-disk layout: summary

- Volume is organized into "quadrangles"
- Parameters: block size, width, and depth
- Disk's physical parameters determine limits
- Quadrangles enable
  - Sequential access for primary axis
  - Semi-sequential access for secondary axis

Atropos abstracts all these details away!
Outline
- Introduction
- Data Layout in DBMS
- Clotho Storage Model
- Atropos Logical Volume Manager
- Experimental Results
- Conclusions

Prototype Implementation
- Shore storage manager [Carey94]
  - Recovery, transaction & concurrency control...
  - NSM, DSM, and PAX layout
- Implement CSM layout & its scan operator
  - Changes within page layout module
  - Less than 1000 lines of code
- Implement new buffer pool manager

Experiments: Shore Storage Mgr
- 4 Seagate Cheetah 36ES 10K RPM disks
- Database workload
  - TPC-C (OLTP workload)
    - Random access to full record
    - Suitable for NSM
  - TPC-H (DSS workload)
    - Sequential access to partial record
    - Suitable for DSM
### Sensitivity Analysis

Table: `CREATE TABLE R (FLOAT a1, ..., FLOAT a15 (1GB))`  
Query: `SELECT a1, a2, ..., FROM R WHERE a1 < Hi`

CSM follows best performance as payload increases

### TPC-H Results

TPC-H: dataset size 1GB  
Sequential partial-record access

<table>
<thead>
<tr>
<th>Query</th>
<th>NSM</th>
<th>DSM</th>
<th>PAX</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>250</td>
<td>200</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>Q6</td>
<td>200</td>
<td>150</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Q12</td>
<td>150</td>
<td>100</td>
<td>50</td>
<td>0</td>
</tr>
</tbody>
</table>

CSM always faster for DSS applications

### TPC-C Result

TPC-C: dataset size 1GB (10 Warehouses, 100 users)  
Random partial-record and full-record access

- CSM runs TPC-C almost as fast as NSM/PAX
- Updates: could do partial record access => higher TpmC
- Scatter-gather I/O => higher TpmC
Data Placement Methods: Comparison

<table>
<thead>
<tr>
<th>Page layout</th>
<th>Cache-memory Performance</th>
<th>Memory-disk Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>full record access</td>
<td>partial record access</td>
</tr>
<tr>
<td>NSM</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>DSM</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>PAX</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>CSM</td>
<td>☺</td>
<td>☺</td>
</tr>
</tbody>
</table>

CSM/Fates: Efficient full- and partial-record accesses at all levels of memory hierarchy

Summary

- Device-independent performance hints
  - encapsulate device-specific characteristics
  - restore assumptions about I/O costs
- Proper division of responsibilities
  - DBMS does not control everything
  - storage subsystem executes I/Os efficiently
- Simplified storage management
  - hands-free performance tuning
- Decoupled data placement for robustness
  - easily add new devices

References


