Physical Data Organization

CPS 216
Advanced Database Systems

Outline

- It’s all about disks
  - That’s why you always draw a database as
- Record layout
- Block layout

Storage hierarchy
How far away is data?

<table>
<thead>
<tr>
<th>Location</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1</td>
</tr>
<tr>
<td>On-chip cache</td>
<td>2</td>
</tr>
<tr>
<td>On-board cache</td>
<td>10</td>
</tr>
<tr>
<td>Memory</td>
<td>100</td>
</tr>
<tr>
<td>Disk</td>
<td>$10^6$</td>
</tr>
<tr>
<td>Tape</td>
<td>$10^9$</td>
</tr>
</tbody>
</table>

(Source: AlphaSort paper, 1995)

A typical disk

Top view

Higher-density sectors on inner tracks and/or more sectors on outer tracks

A block is a logical unit of transfer consisting of one or more sectors
Disk access time

Sum of:
- Seek time: time for disk heads to move to the correct cylinder
- Rotational delay: time for the desired block to rotate under the disk head
- Transfer time: time to read/write data in the block (= time for disk to rotate over the block)

Random disk access

Seek time + rotational delay + transfer time
- Average seek time
  - “Typical” value: 5 ms
- Average rotational delay
  - “Typical” value: 4.2 ms (7200 RPM)

Sequential disk access

Seek time + rotational delay + transfer time
- Seek time
  - (assuming data is on the same track)
- Rotational delay
  - (assuming data is in the next block on the track)
- Easily an order of magnitude faster than random disk access!
Data layout strategy
Keep related things close together!
- Same sector/block
- Same track
- Same cylinder
- Adjacent cylinder

More performance tricks
- Disk scheduling algorithm
  - Example: “elevator” algorithm
- Track buffer
  - Read/write one entire track at a time
- Double buffering
  - While processing the current block in memory, prefetch the next block from disk
- Parallel I/O
  - More disk heads working at the same time

Record layout
Record = row in a table
- Variable-format records
  - Rare in DBMS—table schema dictates the format
  - Maybe relevant for semi-structured data such as XML
- Focus on fixed-format records
  - With fixed-length fields only, or
  - With possible variable-length fields
Fixed-length fields

- All field lengths and offsets are constant
  - Computed from schema, stored in the system catalog
- Example: create table Student(SID integer, name CHAR(20), age integer, GPA float)

\[
\begin{array}{|c|c|c|}
\hline
0 & 4 & 24 & 28 & 36 \\
\hline
142 & Bart (padded with '0') & 10 & 2.3 & \\
\hline
\end{array}
\]

- Watch out for alignment!
- What about NULL?

Variable-length fields

- Example: create table Student
  (SID integer, name VARCHAR(20), age integer, GPA float, comment VARCHAR(100))

- Approach 1: use field delimiters

\[
\begin{array}{|c|c|c|}
\hline
0 & 4 & 8 & 16 \\
\hline
142 & 10 & 2.3 & Bart\texttt{\0} & \texttt{Weird kid}\texttt{\0} \\
\hline
\end{array}
\]

- Approach 2: use an offset array

\[
\begin{array}{|c|c|c|}
\hline
0 & 4 & 8 & 16 & 18 & 22 & 32 \\
\hline
142 & 10 & 2.3 & Bart & \texttt{Weird kid}\texttt{\0} & \\
\hline
22 & 32 & \\
\hline
\end{array}
\]

- Update is messy if it changes the length of a field

LOB fields

- Example: create table Student(SID integer, name CHAR(20), age integer, GPA float, picture BLOB(32000))
- Student records get “de-clustered”
  - Bad because most queries do not involve picture
- Decompose (automatically done by DBMS)
  - Student(SID, name, age, GPA)
  - StudentPicture(SID, picture)
Block layout

How do you organize records in a block?

- NSM (N-ary Storage Model)
  - Most commercial DBMS
- DSM (Decomposition Storage Model)
- PAX (Partition Attributes Across)
  - Recent work (Ailamaki et al., VLDB 2001)

NSM

- Store records from the beginning of each block
- Use a directory at the end of each block
  - To locate records and manage free space
  - Necessary for variable-length records

Options

- Reorganize after every update/delete to avoid fragmentation
- What if records are fixed-length?
  - Reorganize after delete
  - Do not reorganize after update
Cache behavior of NSM

- Query: SELECT SID FROM Student WHERE GPA > 2.0;
- Assumption: cache block size < record size
- Lots of cache misses!
  - Things are not close enough (by memory standard)

Do cache misses matter in DBMS?

- Yes? Percentage of memory-related stall time due to data cache misses:
  - 90% for OLAP workloads
    (lots of large, complex queries; few updates)
  - 50-70% for OLTP workloads
    (lots of small queries and updates)
- No? Compared to disk I/Os, memory-related stall time is nothing

DSM

- Decompose table into smaller ones
  - StudentName(SID, name)
  - StudentAge(SID, age)
  - StudentGPA(SID, GPA)
- Each small table uses NSM
Pros and cons of DSM

Pros

Cons

PAX

- Keep entire rows in a block
- Within a block, cluster columns

<table>
<thead>
<tr>
<th>Bart</th>
<th>Milhouse</th>
<th>Lisa</th>
<th>Ralph</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>3.1</td>
<td>4.3</td>
<td>2.3</td>
</tr>
</tbody>
</table>

PAX versus NSM

- Space requirement
  - Roughly the same
- Cache performance
  - PAX incurs 75% less data cache misses than NSM
- Overall performance
  - For OLAP, PAX is 11-48% faster
  - For OLTP
    - Updates: PAX is 10%-16% faster (assuming NSM reorganizes as well)
    - Queries (typically very selective): I/O still dominates?
Next time

Indexing