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

<table>
<thead>
<tr>
<th>Registers</th>
<th>Cache</th>
<th>Memory</th>
<th>Disk</th>
<th>Tapes</th>
</tr>
</thead>
</table>

How far away is data?

<table>
<thead>
<tr>
<th>Location</th>
<th>Cycles</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1</td>
<td>My head</td>
<td>1 min.</td>
</tr>
<tr>
<td>On-chip cache</td>
<td>2</td>
<td>This room</td>
<td>2 min.</td>
</tr>
<tr>
<td>On-board cache</td>
<td>10</td>
<td>Duke campus</td>
<td>10 min.</td>
</tr>
<tr>
<td>Memory</td>
<td>100</td>
<td>Washington D.C.</td>
<td>1.5 hr.</td>
</tr>
<tr>
<td>Disk</td>
<td>$10^6$</td>
<td>Pluto</td>
<td>2 yr.</td>
</tr>
<tr>
<td>Tape</td>
<td>$10^9$</td>
<td>Andromeda</td>
<td>2000 yr.</td>
</tr>
</tbody>
</table>

(Source: AlphaSort paper, 1995)

I/O dominates—design your algorithms to reduce I/O!

A typical disk

- Tracks
- Platter
- Cylinders
- Arm movement
- Spindle rotation

Rule of thumb: “Moving parts” are slow

Top view

- Higher-density sectors on inner tracks and/or more sectors on outer tracks
- Sectors
- 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
  - Time to skip one half of the cylinders?
  - Not quite; should be time to skip a third of them (why?)
  - “Typical” value: 5 ms
- Average rotational delay
  - Time for a half rotation (a function of RPM)
  - “Typical” value: 4.2 ms (7200 RPM)

Sequential disk access

Seek time + rotational delay + transfer time
- Seek time
  - 0 (assuming data is on the same track)
- Rotational delay
  - 0 (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)

```
<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>24</th>
<th>28</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart (padded with '0')</td>
<td>10</td>
<td>2.3</td>
<td></td>
</tr>
</tbody>
</table>
```

- Watch out for alignment!
  - May need to pad; reorder columns if that helps
- What about NULL?
  - Add a bitmap

**Variable-length fields**

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

```
<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>10</td>
<td>2.3</td>
<td>Bart'0' Weird kid'0'</td>
</tr>
</tbody>
</table>
```

- Approach 1: use field delimiters
  - 0 4 8 16
    - 142 10 2.3 Bart0 Weird kid0

- Approach 2: use an offset array
  - 0 4 8 16 22 32
    - 142 10 2.3 Bart Weird kid

- 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)

**Options**

- Reorganize after every update/delete to avoid fragmentation
  - Need to rewrite half of the block on average
- What if records are fixed-length?
  - Reorganize after delete
    - Only need to move one record
    - Need a pointer to the beginning of free space
  - Do not reorganize after update
    - Need a bitmap indicating which slots are in use

**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

```
| 142 | Bart | 10 | 2.3 |
| 85 | Lisa | 8 | 2.3 |
| 456 | Ralph | 8 | 2.3 |
```

**Examples**

- Reorganize after delete
  - Only need to move one record
  - Need a pointer to the beginning of free space
- Do not reorganize after update
  - Need a bitmap indicating which slots are in use
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)

```
Bart 10 2.3 123 Milhouse 10 3.1
```

```
456 Ralph 8 2.3
```

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

Pros and cons of DSM

Pros
- Smaller records = more will fit in a cache block
- Values in the same column are closer = better locality for queries that just access the key and one column

Cons
- Need to keep the key (or a row ID) in every decomposed table = higher storage overhead
- Different parts of the row are farther apart = bad locality for queries accessing many columns—need joins!

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