Storage

Introduction to Databases
CompSci 316 Fall 2022
Announcements (Thu. Oct 6)

• Next two weeks until 10/20 – project, project, & project!
  • No HW due this week
  • MS-2 due next Thursday (10/13) – see project docs
  • HW4 due 10/20 – to be released next week - group submission per project team on a toy full stack webapp as a warm-up for the project
  • DS6 & DS7 – team work for project & HW4
Where are we now?

Relational model and queries
- Relational Model
- Query in SQL
- Query in RA

Database Design
- E/R diagram (design from scratch)
- Normal Forms (refine design)

Beyond Relational Model
- XML
- NOSQL JSON/MongoDB

DBMS Internals and Query Processing
- Storage
- Index
- Join algo/Sorting
- Execution/Optimization

Transactions
- Basics
- Concurrency Control
- Recovery

(Basic) Big Data Processing
- Map-Reduce
- Parallel DBMS

Covered
Next
To be covered
Why do we draw databases like this?
Outline

• It’s all about disks!
  • That’s why we always draw databases as 
  • And why the single most important metric in database processing is (oftentimes) the number of disk I/O’s performed
Storage hierarchy

Why a hierarchy?
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)

The gap has been widening!

✏️ I/O dominates—design your algorithms to reduce I/O!
Latency Numbers
Every Programmer Should Know

<table>
<thead>
<tr>
<th>Latency Comparison Numbers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 ns</td>
<td></td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
<td></td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 ns</td>
<td></td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 ns</td>
<td></td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
<td></td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000 ns</td>
<td>3 us</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000 ns</td>
<td>10 us</td>
</tr>
<tr>
<td>Read 4K randomly from SSD*</td>
<td>150,000 ns</td>
<td>150 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns</td>
<td>250 us</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000 ns</td>
<td>500 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD*</td>
<td>1,000,000 ns</td>
<td>1,000 us</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns</td>
<td>10,000 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000 ns</td>
<td>20,000 us</td>
</tr>
<tr>
<td>Send packet CA--&gt;Netherlands--&gt;CA</td>
<td>150,000,000 ns</td>
<td>150,000 us</td>
</tr>
</tbody>
</table>

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

Credit
-----
By Jeff Dean: http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers
A typical hard drive

A typical hard drive

“Moving parts” are slow
Top view

“Zoning”: more sectors/data on outer tracks

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

End of Lecture
On 10/6
Disk access time

Sum of:

1. **Seek time**: time for disk heads to move to the correct cylinder
   - Average seek time
     - Sequential: 0
     - Random: “Typical” value: 5 ms

2. **Rotational delay**: time for the desired block to rotate under the disk head
   - Average rotational delay
     - Sequential: 0
     - Random: “Typical” value: 4.2 ms (7200 RPM)

3. **Transfer time**: time to read/write data in the block (= time for disk to rotate over the block)
   - The same for sequential and random

• Sequential is an order of magnitude faster!
Important consequences

• It’s all about reducing I/O’s!

• Cost = #block Read + #block write

• DBMS maintains a memory buffer pool of blocks
  • Typically, << disk blocks

• To read/write any page, a block/page has to be brought to memory from disk (disk read)
  • Some disk blocks may have to evicted to make space for other blocks (replacement strategy: FIFO, LRU, MRU)
  • If a block is Dirty (updated in memory) it must be “flushed” back to disk while being evicted (disk write)
Performance tricks

• Disk layout strategy
  • Keep related things (what are they?) close together: same sector/block → same track → same cylinder → adjacent cylinder

• Prefetching
  • While processing the current block in memory, fetch the next block from disk (overlap I/O with processing)

• Parallel I/O
  • More disk heads working at the same time

• Disk scheduling algorithm
  • Example: “elevator” algorithm

• Track buffer
  • Read/write one entire track at a time
Data layout on disk

How each component is stored in the parent Table → Pages/Blocks → Records/Tuples/Rows → Attributes

Examples:

Fixed-length fields

Variable-length fields (delimiter or offset array)

Not covered in detail

N-ary storage model/NSM
“Row-major”, directory at the end
Reorganization needed after updates

PAX (Partition Attributes Across)
“Column-major” -> Column store
Data layout on disk

How each component is stored in the parent
Table → Pages/Blocks → Records/Tuples/Rows → Attributes
Record layout

Record = row in a table

• Focus on fixed-format records
  • With fixed-length fields only, or
  • With possible variable-length fields

• Variable-format records
  • Rare in DBMS—table schema dictates the format
  • Relevant for semi-structured data such as XML
Fixed-length fields

• All field lengths and offsets are constant
  • Computed from schema, stored in the system catalog

• Example: CREATE TABLE User(uid INT, name CHAR(20), age INT, pop FLOAT);

• Watch out for alignment
  • May need to pad; reorder columns if that helps

• What about NULL?
  • Add a bitmap at the beginning of the record
Variable-length records

• Example: CREATE TABLE User(uid INT, name VARCHAR(20), age INT, pop FLOAT, comment VARCHAR(100));

• Approach 1: use field delimiters (‘\0’ okay?)

• Approach 2: use an offset array

• Put all variable-length fields at the end (why?)
• Update is messy if it changes the length of a field
LOB fields

• ”Large Object” column (max 4 GB in Oracle)

• Example: CREATE TABLE User(uid INT,
   name CHAR(20), age INT,
   pop FLOAT, picture BLOB(32000));

• Student records get “de-clustered”
  • Bad because most queries do not involve picture

• Decomposition (automatically and internally done by DBMS without affecting the user)
  • (uid, name, age, pop)
  • (uid, picture)
Block layout

How do you organize records in a block?

• **NSM** (N-ary Storage Model)
  • Most commercial DBMS
  • “Row-major”

• **PAX** (Partition Attributes Across)
  • Ailamaki et al., VLDB 2001
  • “Column-major”
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

Why store data and directory at two different ends?

So both can grow easily!
Options

• Reorganize after every update/delete to avoid fragmentation (gaps between records)
  • Need to rewrite half of the block on average

• A special case: What if records are fixed-length?
  • Option 1: reorganize after delete
    • Only need to move one record
    • Need a pointer to the beginning of free space
  • Option 2: do not reorganize after update
    • Need a bitmap indicating which slots are in use
Cache behavior of NSM

- **Query:** SELECT uid FROM User WHERE pop > 0.8;
- **Assumptions:** no index, and cache line size < record size
- **Lots of cache misses**
  - uid and pop are not close enough by memory standards

```
142  Bart    10  0.9
857  Lisa    8   0.7
123 Milhouse 10  0.2
456  Ralph   8   0.3
```
PAX (column major)

- Most queries only access a few columns
- Cluster values of the same columns in each block
  - When a particular column of a row is brought into the cache, the same column of the next row is brought in together

Reorganize after every update (for variable-length records only) and delete to keep fields together

(IS NOT NULL bitmap)
Beyond block layout: column stores

• The other extreme: store tables by columns instead of rows

• Advantages (and disadvantages) of PAX are magnified
  • Not only better cache performance, but also fewer I/O’s for queries involving many rows but few columns
  • Aggressive compression to further reduce I/O’s

• More disruptive changes to the DBMS architecture are required than PAX
  • Not only storage, but also query execution and optimization

• Example: Apache Parquet
Take Away

• Storage hierarchy
  • Why I/O’s dominate the cost of database operations

• Disk
  • Steps in completing a disk access
  • Sequential versus random accesses

• Record layout
  • Handling variable-length fields
  • Handling NULL
  • Handling modifications

• Block layout
  • NSM: the traditional layout
  • PAX: a layout that tries to improve cache performance

• Column stores: NSM transposed, beyond blocks

---

Disk is slower than Main memory = Buffer Pool
Minimize the number of transfers to/from Disk
Our unit of cost!
All computation cost ignored by default – unless mentioned otherwise