CS 133: Databases

Fall 2016
Lec 02 – 9/5
Prof. Beth Trushkowsky

Goals for Today

• Understand the storage hierarchy and why disk input/output (I/O) is an important metric for query cost

• See how different policies for managing which data stays in RAM can impact cost of queries

Administrivia

• Problem set 1 out, due Thursday 11:59pm
  – Honor code: can use lectures notes and textbook, can discuss general ideas with classmates
  – On Sakai

• Lab 1: Getting started due Wednesday
  – On course website
    (linked from assignment on Sakai)

Relational Model

• Users write declarative queries using logical schema
  – May actually interact with application that queries the database
  – Database administrator (DBA) typically creates database

• Given declarative query, DBMS figures out efficient execution strategy

We’ll start discussion of “choices” today!

Courses (cid: string, name: string, credits: integer)
Simplified RDBMS Architecture

Let’s look at the system bottom-up!

Query
  → Query optimizer
  → Access methods
  → Buffer management
  → Disk management

Data records

Query results

Query execution

Concerned with concurrency control and recovery

Computer Storage

**Primary storage**
- E.g., “main memory” a.k.a random-access memory (RAM)
- Typically volatile

**Secondary storage**
- Hard disk drive
- Non-volatile

Why Not Keep All Data in Memory?

- Costs too much!
  - $100 for 100 GB of RAM or around 2 TB of disk
  - Databases can be in the petabyte (1000 TB) range

- Main memory volatile
  - Want persistence

A Typical Disk

- Moving parts!
  - Platters spin
  - Arms move in/out to position heads with track
  - Tracks under heads make conceptual cylinder

- A *block* is a unit of transfer
  - made up of one or more sectors

In main memory, we’ll call this chunk of data a *page*
Disk Access Time

- Time to read/write (an Input/Output or I/O) a block
  - Seek time
  - Rotational Delay
  - Transfer time

- Seek time and rotational delay dominate (stats: wikipedia)
  - Seek time: about 4 to 15msec
  - Rotational: avg 4msec (7200rpm)
  - Transfer rate: < 0.1msec per 8KB block

Reduce I/O cost by reducing seek and rotation

Random vs. Sequential Access

If data can be read/written sequentially, have zero seek time and rotational delay!

Exercise 3: Counting I/Os

- Query: joining relations Students and Enrolled
  
  SELECT S.name, E.CID
  FROM Students S, Enrolled E
  WHERE S.sid=E.sid;

- [Simple] join pseudocode:
  
  For each tuple i of outer relation
  For each tuple j of inner relation
  Check if i.sid == j.sid

- Relation info
  - Students: 20 pages, 1000 total tuples
  - Enrolled: 50 pages, 6000 total tuples
  - For a given relation, pages on disk sequentially
Exercise: Counting I/Os

- Think of the simple algorithm as a nested for-loop like this:

For each page of \textit{Outer} relation
  Load that page // one I/O

For each tuple of \textit{Outer} on that page
  For each page of \textit{Inner} relation
    Load that page // one I/O
    For each tuple of \textit{Inner} on that page
      // do tuple comparison

- Total I/Os = (\# pages in \textit{outer}) + (\# tuples in \textit{outer}) * (\# pages in \textit{inner})
  - Students outer: 20 + 1000*50 = 50,020
  - Enrolled outer: 50 + 6000*20 = 120,050

- \# Random I/Os = (\# pages in \textit{Outer}) + (\# tuples in \textit{Outer})(1)

- \# Sequential I/Os = (\# pages in \textit{Inner} – 1) (\# tuples in \textit{Outer})

Exercise: Counting I/Os

Simplified RDBMS Architecture

- Disk Space Manager
  - Manages space on disk
  - Higher levels call on it to allocate/de-allocate, and read/write \textit{pages}

The Buffer Manager

- Data must be RAM for DBMS to operate on it
  - Too costly to keep all data in RAM

- Buffer manager
  - Maintain a pool of space in RAM
  - Talks to disk space manager to read/write pages
  - Higher levels do not know what is in RAM or not
Buffer Pool

When a Request Comes in...

- If requested page is in the buffer pool
  - Pin the page to mark as in use
- Else, if requested page is not in buffer pool
  - If there is an available frame, put the page in that frame
    - Else, select a frame for replacement using a replacement policy
      (only un-pinned pages are eligible for replacement)
      - If selected frame is dirty, write it back to disk
      - Read requested page into the selected frame
      - Pin the page

Buffer Replacement Policy

- When no available frames in buffer pool, need to evict one based on a replacement policy
  - Choice of policy impacts number of disk I/Os
  - Efficacy depends on access pattern of pages

What would an optimal policy do?

Important Terms

- Disk page: unit of transfer between disk and memory. Size is DBMS configuration parameter (e.g., 4-32 KB).
- Frame: unit of memory. Typically same size as disk page size.
- Buffer Pool: collection of frames used to temporarily keep data for query processor.
LRU Policy (Least Recently Used)

- Evict the page that was accessed (pinned) furthest in the past, i.e., the least recently used of the pages in the pool

- Example:
  - Buffer pool with 4 frames
  - Assume pages are immediately unpinned after use

Access pattern:

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Intuition: if a page has not been used in a while, it probably won’t again soon

# hits: 2
# misses: 6

Issues with LRU

- Sequential flooding
  - # buffer frames < # pages in file
  - each request causes an I/O

  - E.g., repeated sequential scans

  - MRU (most recently used) policy can be a better policy in this case