Storing Data: Disks and Files

Storing and Retrieving Data

- Database Management Systems need to:
  - Store large volumes of data
  - Store data reliably (so that data is not lost!)
  - Retrieve data efficiently
- Alternatives for storage
  - Main memory
  - Disks
  - Tape

Why Not Store Everything in Main Memory?

- Costs too much. $100 will buy you either 2GB of RAM (similar for flash memory) or 400GB of disk today.
- Main memory is volatile. We want data to be saved between runs. (Obviously!)
  - Flash memory is non-volatile
**Why Not Store Everything in Tapes?**

- **No random access.** Data has to be accessed sequentially
  - Not a great idea when accessing a small portion of a terabyte of data
- **Slow!** Data access times are larger than for disks

**Disks**

- **Secondary storage device of choice**
  - Cheap
  - Stable storage medium
  - Random access to data
- **Main problem**
  - Data read/write times much larger than for main memory
  - Positioning time in order of milliseconds
    - How many instructions could a 3 GHz CPU process during that time…

**Solution 1: Techniques for making disks faster**

- **Intelligent data layout on disk**
  - Put related data items together
- **Redundant Array of Inexpensive Disks (RAID)**
  - Achieve parallelism by using many disks
Solution 2: Buffer Management

- Keep “currently used” data in main memory
  - How do we do this efficiently?
- Typical (simplified) storage hierarchy:
  - Main memory (RAM) for currently used data
  - Disks for the main database (secondary storage)
  - Tapes for archiving older versions of the data (tertiary storage)

Outline

- Disk technology and how to make disk read/writes faster
- Buffer management
- Storing “database files” on disk

Components of a Disk

- The platters spin (say, 10K rpm).
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a cylinder (imaginary!).
- Only one head reads/writes at any one time.
- Block size is a multiple of sector size (which is fixed).
**Accessing a Disk Page**

- Time to access (read/write) a disk block:
  - **seek time** (moving arms to position disk head on track)
  - **rotational delay** (waiting for block to rotate under head)
  - **transfer time** (actually moving data to/from disk surface)

- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 0.1-0.5msec per 4KB page

- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

**Arranging Pages on Disk**

- `Next` block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder

- Blocks in a file should be arranged sequentially on disk (by `next`), to minimize seek and rotational delay.

- For a sequential scan, pre-fetching several pages at a time is a big win!

**RAID**

- **Redundant Array of Inexpensive Disks**
  - A.k.a. Redundant Array of Independent Disks

- **Disk Array**: Arrangement of several disks that gives abstraction of a single, large disk.

- **Goals**: Increase performance and reliability.

- **Two main techniques**:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks -> more failures. Redundant information allows reconstruction of data if a disk fails. Two main approaches: parity and mirroring.
**Parity**

- Add 1 redundant block for every n blocks of data
  - XOR of the n blocks
- Example: D1, D2, D3, D4 are data blocks
  - Compute DP as D1 XOR D2 XOR D3 XOR D4
  - Store D1, D2, D3, D4, DP on different disks
  - Can recover any one of them from the other four by XORing them

**RAID Levels**

- Level 0: No redundancy
  - Striping without parity
- Level 1: Mirrored (two identical copies)
  - Each disk has a mirror image (check disk)
  - Parallel access: reduces positioning time, but transfer only from one disk.
  - Maximum transfer rate = transfer rate of one disk
  - Write involves two disks.

**RAID Levels (Contd.)**

- Level 0+1: Striping and Mirroring
  - Parallel reads
  - Write involves two disks
  - Maximum transfer rate = aggregate bandwidth
  - Combines performance of RAID 0 with redundancy of RAID 1.
- Example: 8 disks
  - Divide into two sets of 4 disks
  - Each set is a RAID 0 array
  - One set mirrors the other
RAID Levels (Contd.)

- Level 3: Bit-Interleaved Parity
  - Striping Unit: One bit. One check disk.
  - Each read and write request involves all disks; disk array can process one request at a time.

RAID Levels (Contd.)

- Level 4: Block-Interleaved Parity
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests, large requests can utilize full bandwidth
  - Writes involve modified block and check disk

RAID Levels (Contd.)

- Level 5: Block-Interleaved Distributed Parity
  - Similar to RAID Level 4, but parity blocks are distributed over all disks
  - Eliminates check disk bottleneck, one more disk for higher read parallelism
In-Class Exercise

- How does the striping granularity (size of a stripe) affect performance, e.g., RAID 3 vs. RAID 4?
  - Smaller stripe -> file is broken into more and smaller pieces -> small files are distributed over more disks -> faster transfer when reading that file (parallel I/O)
  - Disadvantage: when reading multiple files, each disk has more requests, leading to worse positioning time (seek + rotational delay)
  - Write performance: need not (!) read whole stripe to re-compute parity
    - NewParity = (OldData XOR NewData) XOR OldParity

Which RAID to Choose?

- RAID 0: great performance at low cost, limited reliability
- RAID 0+1 (better than 1): small storage subsystems (cost of mirroring limited), or when write performance matters
- RAID 3 (better than 2): large transfer requests of contiguous blocks, bad for small requests of single blocks
- RAID 5 (better than 4): good general-purpose solution
Which RAID to Choose? Corrected.

- RAID 0: great performance at low cost, limited reliability
- RAID 0+1 (better than 1): small storage subsystems (cost of mirroring limited), or when write performance matters
- RAID 5 (better than 3, 4): good general-purpose solution

RAID Comparison

This is just a rule-of-thumb comparison: don’t worry about half a star difference, RAID 3 is overrated etc.

Disk Space Management

- Lowest layer of DBMS software manages space on disk.
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Request for a sequence of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don’t need to know how this is done, or how free space is managed.
Outline

- Disk technology and how to make disk read/writes faster
- Buffer management
- Storing “database files” on disk

Buffer Management in a DBMS

- Data must be in RAM for DBMS to operate on it!
- Table of \(<\text{frame}\#, \text{pageid}\>\) pairs is maintained.

When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for replacement
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- Pin the page and return its address.

- If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!
More on Buffer Management

- Requestor of page must unpin it, and indicate whether page has been modified:
  - dirty bit is used for this.
- Page in pool may be requested many times,
  - a pin count is used. A page is a candidate for replacement iff pin count = 0.
- CC & recovery may entail additional I/O when a frame is chosen for replacement.
  (Write-Ahead Log protocol; more later.)

In Class Exercise

- What happens if the buffer is full and all frames have pin count > 0?
- What happens if multiple transactions (users) want to access the same page?

Buffer Replacement Policy

- Frame is chosen for replacement by a replacement policy:
  - Least-recently-used (LRU): priority queue based on last access to frame (time when pin count goes to 0)
  - Clock: round-robin replacement with referenced bit
  - Many others
    - First-in-first-out (FIFO), Most-recently-used (MRU), Random
**Buffer Replacement Policy (Contd.)**

- Policy can have big impact on # of I/O's; depends on the access pattern.
- **Sequential flooding**: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O.
  - Which replacement policy is better?

**DBMS vs. OS File System**

- OS does disk space & buffer mgmt: why not let OS manage these tasks?
- Differences in OS support: portability issues
- Some limitations, e.g., files can’t span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust replacement policy, and pre-fetch pages based on access patterns in typical DB operations.

**Outline**

- Disk technology and how to make disk read/writes faster
- Buffer management
- Storing “database files” on disk
Files of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on records, and files of records.
- **FILE**: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - read a particular record (specified using record id)
  - scan all records (possibly with some conditions on the records to be retrieved)

Record Formats: Fixed Length

- Information about field types same for all records in a file; stored in system catalogs.
- Finding i'th field requires scan of record.

Record Formats: Variable Length

- Two alternative formats (# fields is fixed):
  - Second offers direct access to i'th field, efficient storage of nulls (special don't know value); small directory overhead.
### Page Formats: Fixed Length Records

- **Slot 1**
- **Slot 2**
- **Slot N**

**Free Space**

**Number of records**

- **Packed**
- **Free Space**

### Page Formats: Fixed Length Records

- **Slot 1**
- **Slot 2**
- **Slot N**

**Free Space**

**Number of slots**

- **Unpacked, Bitmap**

### Page Formats: Variable Length Records

- **Rid = (i, N)**
- **Rid = (i, 2)**
- **Rid = (i, 1)**

**Pointer to start of free space**

**SLOT DIRECTORY**

N . . . 24 16 2 1

**Number of slots**

* Can move records on page without changing rid; so, attractive for fixed-length records too.*
**Unordered (Heap) Files**

- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the pages in a file
  - keep track of free space on pages
  - keep track of the records on a page
- There are many alternatives for keeping track of this.

**Heap File Implemented as a List**

- The header page id and Heap file name must be stored someplace.
- Each page contains 2 'pointers' plus data.

**Heap File Using a Page Directory**

- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative.
  - Much smaller than linked list of all HF pages!
Indexes

- A Heap file allows us to retrieve records:
  - by specifying the rid
  - Usually <page id, slot number>, or some integer (need lookup table for corresponding page id and slot number)
  - by scanning all records sequentially
- Sometimes, we want to retrieve records by specifying the values in one or more fields, e.g.,
  - Find all CS students with a gpa > 3
- Indexes are file structures that enable us to answer such value-based queries efficiently.

System Catalogs

- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

* Catalogs are themselves stored as relations!

<table>
<thead>
<tr>
<th>Attribute_Name</th>
<th>Relation_Name</th>
<th>Type</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>Students</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>Students</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>login</td>
<td>Students</td>
<td>string</td>
<td>3</td>
</tr>
<tr>
<td>age</td>
<td>Students</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>gpa</td>
<td>Students</td>
<td>real</td>
<td>5</td>
</tr>
<tr>
<td>fid</td>
<td>Faculty</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>fname</td>
<td>Faculty</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>sal</td>
<td>Faculty</td>
<td>real</td>
<td>3</td>
</tr>
</tbody>
</table>
Summary

- Disks provide cheap, non-volatile storage
- Buffer manager brings pages into RAM
- DBMS vs. OS File Support
- Fixed and Variable length records
- Slotted page organization