CSE232: Database System Principles

Hardware
Database System Architecture

Query Processing

- SQL query
- Parser
- relational algebra
- Query Rewriter and Optimizer
- Query execution plan
- Execution Engine

Transaction Management

- Calls from Transactions (read, write)
- Transaction Manager
- Recovery Manager
- Log
- Hardware aspects of storing and retrieving data
- Lock Table

View definitions
Statistics & Catalogs & System Data

Data + Indexes
Memory Hierarchy

• Cache memory
  – On-chip and L2
  – Caching outside control of DB system, still good to know

• RAM (controlled by db system)
  – Addressable space includes virtual memory but DB systems avoid it

• SSDs
  – Block-based storage

• Disk
  – Block + preference to sequential access

• Tertiary storage for archiving
  – Tapes, jukeboxes, DVDs
  – Does not matter any more
Volatile Vs Non-Volatile Storage

- Persistence important for transaction atomicity and durability
- Even if database fits in main memory changes have to be written in non-volatile storage
- Hard disk
- RAM disks w/ battery
- Flash memory
Peculiarities of storage mediums affect algorithm choice

• Block-based access:
  – Access performance: How many blocks were accessed
  – How many objects
  – Flash is different on reading Vs writing

• Clustering for sequential access:
  – Accessing consecutive blocks costs less on disk-based systems
Moore’s Law: Different Rates of Improvement Lead to Algorithm & System Reconsiderations

- Processor speed
- Main memory bit/$
- Disk bit/$
- RAM access speed
- Disk access speed
- Disk transfer rate

Clustered/sequential access-based algorithms became relatively better
Moore’s Law: Same Phenomenon Applies to RAM

Algorithms that access memory sequentially have better constant factors than algorithms that access randomly.
Moore’s Law: Different Rates of Improvement => Different Buffering Considerations

Cost of “miss” increases

- Cache Capacity
- RAM Capacity
- Disk Access Speed
Case Study: Disk

Terms: Platter, Head, Actuator Cylinder, Track Sector (physical), Block (logical), Gap
**Top View**

Often different numbers of sectors per track

- **Sector**
- **Track**
- **Gap**
- **Block** (typically multiple sectors)
<table>
<thead>
<tr>
<th>Property</th>
<th>“Typical” Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter</td>
<td>1 inch → 15 inches</td>
</tr>
<tr>
<td>Cylinders</td>
<td>100 → 20000</td>
</tr>
<tr>
<td>Surfaces</td>
<td>1 (CDs) → 2 (floppies) → 5 (typical hd) → 30</td>
</tr>
<tr>
<td>(Tracks/cyl)</td>
<td></td>
</tr>
<tr>
<td>Sector Size</td>
<td>512B → 50K</td>
</tr>
<tr>
<td>Capacity</td>
<td>360 KB (old floppy) → 4 TB</td>
</tr>
</tbody>
</table>
Key performance metric: Time to fetch block

I want block X in memory

Time = Seek Time (locate track) + Rotational Delay (locate sector) + Transfer Time (fetch block) + Other (disk controller, ...)

?
Seek Delay

Track Where Head must go

Track Where Head is
Rotational Delay

Head Here

Block I Want
Seek Time

3 or 5x

Time

Few ms

Cylinders Traveled
Average Random Seek Time

\[ S = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \text{SEEKTIME} (i \rightarrow j)}{N(N-1)} \]

“Typical” S: 10 ms
Average Rotational Delay

R = 1/2 revolution

“typical” R = 8.33 ms (7200 RPM)

Assume we have to start reading from start of first sector
Transfer Rate: $t$

- “typical” $t$: $1 \rightarrow 1-2$ Gbits/second
- transfer time: \underline{block size} $t$
Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

“Typical” Value: 0
Practice Problem

• Single surface
• Rotation speed 7200rpm
• 16,384 tracks
• 128 sectors/track
• 4096 bytes/sector
• 4 sectors/block (16,384 bytes/block)
• SEEKTIME (i → j) = [1000 + (j-i)] μs
• Neglect gaps
• Calculate minimum, maximum, average time to fetch one block
Practice Problem: Minimum Time

• Head is at the start of the first sector of the block
• Just compute transfer time
• 4 sectors cover 4/128 of a track
• 1 full rotation takes 60/7200=8.33ms
• Transfer time is 8.33 * 4 /128 = 0.26ms
Practice Problem: Maximum Time

• Assume read must start at the first sector
• Head is at innermost, required track is the outermost
• Seek time = ...
• Head just missed the beginning
• Rotational delay = ...
• Transfer time = ...
Practice problem: Average time

• Solve...
• So far: Random Block Access
• What about: Reading “Next” block?

\[
\text{Time to get} = \text{Block Size} + \text{Negligible } t
\]
- skip gap
- switch track
- once in a while, next cylinder
<table>
<thead>
<tr>
<th>Rule of Thumb</th>
<th>Random I/O: Expensive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential I/O: Much less</td>
</tr>
</tbody>
</table>
Practice Problem cont’d: Sustained Bandwidth over Track

- Assume required blocks are consecutive on single track
- What is the sustained bandwidth of fetching consecutive blocks?
  - 128 sectors/track * 4KB/sector in 8.33ms/track full rotation = 512KB/8.33ms = 61.46KB/ms
Suggested optimization

• Cluster data in consecutive blocks
• Give an extra point to algorithms that
  – exploit data clustering by avoiding “random” accesses
  – Read/write consecutive blocks
2-Phase Merge Sort: An algorithm tuned for blocks and sequential access

Assume a file with many records. Each record has a key and other data. For ppt brevity, the slide shows only the key of each record and not its data. Assume each block has 2 records. Assume RAM buffer fits 4 blocks (8 records). In practice, expect many more records per block and many more records fitting in buffer.

**Problem:** Sort the records according to the key.

**Morale:** What you learnt in algorithms and data structures is not always the best when we consider block-based storage.
2-Phase Merge Sort: An algorithm tuned for blocks and sequential access

Phase 1, round 1

READ

PKADLEZW JCRHYFXI

RAM buffer

Secondary storage

SORT in place, eg quicksort

WRITE

ADEKLPWZ

ADEKLPWZ
2-Phase Merge Sort: An algorithm tuned for blocks and sequential access

Phase 1, round 2
Phase 2 continues until no more records

Secondary storage

1st file

2nd file

In practice, probably many more Phase 1 rounds and many respective output files
2-Phase Merge Sort: An algorithm tuned for blocks and sequential access

Phase 2

Assume #files < #blocks that fit in RAM buffer.
Fetch the first block of each file in RAM buffer.
Merge records and output.
When all records of a block have been output, bring next block of same file

Improvement: Bring max number of blocks in memory.
2-Phase Merge Sort: Most files can be sorted in just 2 passes!

Assume

- $M$ bytes of RAM buffer (eg, 8GB)
- $B$ bytes per block (eg, 64KB for disk, 4KB for SSD)

Calculation:
- The assumption of Phase 2 holds when $\#files < \frac{M}{B}$
  => there can be up to $\frac{M}{B}$ Phase 1 rounds
- Each round can process up to $M$ bytes of input data
  => 2-Phase Merge Sort can sort $\frac{M^2}{B}$ bytes

  - eg $(8\text{GB})^2/64\text{KB} = (2^{33}\text{B})^2 / 2^{16}\text{B} = 2^{50}\text{B} = 1\text{PB}$
Horizontal placement of SQL data in blocks

Relations:
• Pack as many tuples per block
  – improves scan time
• Do not reclaim deleted records
• Utilize overflow records if relation must be sorted on primary key
• A novel generation of databases features column storage
  – to be discussed late in class
Pack maximum # records per block

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>number</th>
<th>date_code</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web</td>
<td>CSE135</td>
<td>TuTh</td>
<td>2:00</td>
<td>3:20</td>
</tr>
<tr>
<td>2</td>
<td>Databases</td>
<td>CSE132A</td>
<td>TuTh</td>
<td>3:30</td>
<td>4:50</td>
</tr>
<tr>
<td>4</td>
<td>VLSI</td>
<td>CSE121</td>
<td>F</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

If hard drive disk: Preferably store in consecutive blocks or sequentially in file

“pack” each block with maximum # records
Utilize overflow blocks for insertions with “out of order” primary keys

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>number</th>
<th>date_code</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web</td>
<td>CSE135</td>
<td>TuTh</td>
<td>2:00</td>
<td>3:20</td>
</tr>
<tr>
<td>2</td>
<td>Databases</td>
<td>CSE132A</td>
<td>TuTh</td>
<td>3:30</td>
<td>4:50</td>
</tr>
<tr>
<td>3</td>
<td>PL</td>
<td>CSE130</td>
<td>TuTh</td>
<td>9:00</td>
<td>9:50</td>
</tr>
<tr>
<td>4</td>
<td>VLSI</td>
<td>CSE121</td>
<td>F</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Classes

1. Web CSE135 TuTh 2:00 3:20
2. Databases CSE132A TuTh 3:30 4:50
3. PL CSE130 TuTh 9:00 9:50
4. VLSI CSE121 F null

Overflow block

just inserted tuple

just inserted tuple

Overflow block
Block Size Selection?

- Big Block $\rightarrow$ Amortize I/O Cost

Unfortunately...

- Big Block $\Rightarrow$ Read in more useless stuff! and takes longer to read
Trend

- memory prices drop and memory capacities increase,
- transfer rates increase
- disk access times do not reduce that much

⇒ blocks get bigger