CS4432: Database Systems II

Lecture #3

Professor Elke A. Rundensteiner
Quick Logistics
Where have I been?
(at EDBT’04 in sunny Greece)

Email backlog ...
• Use **MyWPI** for general questions
• Use [rundenst@cs.wpi.edu](mailto:rundenst@cs.wpi.edu) for specific questions to me only; but make sure to have “CS4432” in email header
• BS/MS Credit for this class (talk to me)
• MQPs in DB for next year (talk to me)
• I’ll stay after class today to address anything that needs immediate attention.
Lecture #3

Still on Chapter 2 (in textbook)

Today : On Disk Optimizations
Next Week: On Storage Layout
Thus far:

- Hardware: Disks
- Architecture: Layers of Access
- Access Times and Abstractions
- Example - Megatron 747 from textbook
TODAY:

• Using secondary storage effectively (Sec. 2.3)

• SKIP part of chapter 2: Disk Failure Issues
One Simple Idea: Prefetching

Problem: Have a File
  » Sequence of Blocks B1, B2

Have a Program
  » Process B1
  » Process B2
  » Process B3
  »
Single Buffer Solution

(1) Read B1 → Buffer
(2) Process Data in Buffer
(3) Read B2 → Buffer
(4) Process Data in Buffer ...
Say \( P = \) time to process/block
\( R = \) time to read in 1 block
\( n = \) # blocks

Single buffer time = \( n(P+R) \)
Question:
Could the DBMS know something about behavior of such future block accesses?

What if:
If we knew more about sequence of future block accesses, what and how could we do better?
Idea: Double Buffering/Prefetching

Memory:

Disk:

process

process

C

B

don
done

C  D  E  F  G
Say $P \geq R$

$P = \text{Processing time/block}$
$R = \text{IO time/block}$
$n = \# \text{ blocks}$

What is processing time now?

- Double buffering time $= ?$
Say $P \geq R$

$P = \text{Processing time/block}$

$R = \text{IO time/block}$

$n = \text{# blocks}$

- Double buffering time $= R + nP$

- Single buffering time $= n(R+P)$
Block Size Selection?

• **Question:**
  Do we want Small or Big Block Sizes?

• Pros?
• Cons?
Block Size Selection?

- Big Block → Amortize I/O Cost
  - For seek and rotational delays are reduced ...

Unfortunately...

- Big Block ⇒ Read in more useless stuff! and takes longer to read
Trend

- As memory prices drop, blocks get bigger ...
Using secondary storage effectively

- Example: Sorting data on disk
- General Wisdom:
  - I/O costs dominate
  - Design algorithms to reduce I/O
Disk IO Model Of Comptations

→

Efficient Use of Disk

Example: Sort Task
“Good” DBMS Algorithms

- Try to make sure if we read a block, we use much of data on that block
- Try to put blocks together that are accessed together
- Try to buffer commonly used blocks in main memory
Why Sort Example?

- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing gpa order
- Sorting is first step in bulk loading B+ tree index.
- Sorting useful for eliminating duplicate copies in a collection of records (Why?)
- Sort-merge join algorithm involves sorting.
- Problem: sort 1Gb of data with 1Mb of RAM.
  - why not virtual memory?
Sorting Algorithms

• Any examples algorithms you know ??
• Typically they are main-memory oriented
• They don’t look too good when you take disk I/Os into account ( why? )
Merge Sort

- Merge: Merge two sorted lists and repeatedly choose the smaller of the two “heads” of the lists.

- Merge Sort: Divide records into two parts; merge-sort those recursively, and then merge the lists.
2-Way Sort: Requires 3 Buffers

• Pass 1: Read a page, sort it, write it.
  - only one buffer page is used

• Pass 2, 3, ..., etc.:
  - three buffer pages used.

Disk → INPUT 1 → OUTPUT → Disk
       \  /       /  \\
       INPUT 2 →       \\
Main memory buffers
Two-Way External Merge Sort

• Idea:  *Divide and conquer:* sort subfiles and merge
Two-Way External Merge Sort

- What is the total cost for sorting?
- How many passes do we need?
- Costs for each pass?

Input file

1-page runs

2-page runs

4-page runs

8-page runs

PASS 0

PASS 1

PASS 2

PASS 3

9
3,4 6,2 9,4 8,7 5,6 3,1 2
3,4 5,6 2,6 4,9 7,8 1,3 2
2,3 4,6
4,7 8,9
1,3 5,6 2
3,5
2
6

1.2

6.7

6.6

3.4

3.4

1.2

4.4

2.3

4.6

8.9

1.2

6

4.4

4.6

8.9

1.3

5.6

1.3

5.6

1.3

5.6

2

2
Two-Way External Merge Sort

- Each pass we read + write each page in file.
  \[ = 2 \times N \]

- \( N \) pages in file \( \Rightarrow \) number of passes:
  \[ = \lceil \log_2 N \rceil + 1 \]

- So total cost is:
  \[ 2N(\lceil \log_2 N \rceil + 1) \]
General External Merge Sort

• What if we had more buffer pages?
• How do we utilize them?
General External Merge Sort

Input \( N \) pages using \( B \) buffer pages?
General External Merge Sort

- To sort file with $N$ pages using $B$ buffer pages
- Phase 1 (pass 0):
  - Fill memory with records
  - Sort using any favorite main-memory sort
  - Write sorted records to disk
  - Repeat above, until all records have been put into one sorted list
General External Merge Sort

• Phase 1 (pass 0): using $B$ buffer pages
  - Produce what output ???
  - Cost (in terms of I/Os) ???

---

Diagram:

- Disk
- INPUT 1
- INPUT 2
- INPUT B
- B Main memory buffers
- Disk
General External Merge Sort

• To sort file with $N$ pages using $B$ buffer pages:
  – **Produce output**: Sorted runs of $B$ pages each
    • Run Sizes: $B$ pages each run.
    • How many runs: $\lceil N / B \rceil$ runs.
  – **Cost**: ?
General External Merge Sort

- To sort file with \( N \) pages using \( B \) buffer pages:
  - Pass 0: use \( B \) buffer pages.
  - Produce output: Sorted runs of \( B \) pages each
    - Run Sizes: \( B \) pages each run.
    - How many runs: \( \lceil N / B \rceil \) runs.
  - Cost:
    - \( 2 * N \) I/Os
General External Merge Sort

- Sort $N$ pages using $B$ buffer pages:
  - Phase 1 (which is pass 0).
    Produce sorted runs of $B$ pages each.
  - Phase 2 (may involve several passes 2, 3, etc.)
    Each pass merges $B - 1$ runs.
Phase 2

- Initially load input buffers with the first blocks of respective sorted run
- Repeatedly run a competition among list unchosen records of each of buffered blocks
  - Move record with least key to output
- Manage buffers as needed:
  - If input block exhausted, get next block from file
  - If output block is full, write it to disk
General External Merge Sort

- Sort $N$ pages using $B$ buffer pages:
  - Phase 1 (which is pass 0).
    Produce sorted runs of $B$ pages each.
  - Phase 2 (may involve several passes 2, 3, etc.)

Number of passes? Cost of each pass?

![Diagram of General External Merge Sort]

**Diagram:**
- Disk
  - INPUT 1
  - INPUT 2
  - INPUT B-1
- B Main memory buffers
- OUTPUT
- Disk
Cost of External Merge Sort

- **Number of passes:** \(1 + \left\lceil \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil\)
- **Cost** = \(2N \times \text{(Number of passes)}\)
- **Total Cost**: multiply above
Example

• Buffer : with 5 buffer pages,
• File to sort : 108 pages

– Pass 0:
  • Size of each run?
  • Number of runs?

– Pass 1:
  • Size of each run?
  • Number of runs?

– Pass 2:  ???
Example

• Buffer: with 5 buffer pages
• File to sort: 108 pages
  - Pass 0: \( \lceil 108 / 5 \rceil = 22 \) sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: \( \lceil 22 / 4 \rceil = 6 \) sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

• Total I/O costs: ?
Example

- Buffer: with 5 buffer pages
- File to sort: 108 pages
  - Pass 0: \( \lceil 108 / 5 \rceil = 22 \) sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: \( \lceil 22 / 4 \rceil = 6 \) sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

- Total I/O costs: \( 2\times N (4) \)
## Number of Passes of External Sort

<table>
<thead>
<tr>
<th>N</th>
<th>B=3</th>
<th>B=5</th>
<th>B=9</th>
<th>B=17</th>
<th>B=129</th>
<th>B=257</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10,000</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100,000</td>
<td>17</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,000,000</td>
<td>20</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10,000,000</td>
<td>23</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>26</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>30</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
How large a file can be sorted in 2 passes with a given buffer size M?
Double Buffering (Useful here)

- To reduce wait time for I/O request to complete, can `prefetch` into `shadow block`.
  - Potentially, more passes; in practice, most files *still* sorted in 2 or at most 3 passes.
Sorting Summary

• External sorting is important; DBMS may dedicate part of buffer pool for sorting!

• External merge sort minimizes disk I/O cost
  – Larger block size means less I/O cost per page.
  – Larger block size means smaller # runs merged

• In practice, # of runs rarely > 2 or 3
Recap

Today : On Disk Optimizations
Next Week: On Storage Layout

Start to read chapter 3 in textbook
Homework 1

Out: Today
Due: Next Friday (in class)