EPL446 – Advanced Database Systems

Lecture 9

External Sorting

Chapter 13: Ramakrishnan & Gehrke

Demetris Zeinalipour

http://www.cs.ucy.ac.cy/~dzeina/courses/epl446
External Sorting Introduction
(Εξωτερική Ταξινόμηση: Εισαγωγή)

• **Problem:** We can’t sort 1TB of data with 1GB of RAM (i.e., more data than available memory) in main memory

• **Solution:** Utilize an External Sorting Algorithm
  – **External sorting** refers to the sorting of a file that resides on secondary memory (e.g., disk, flash, etc).
  – **Internal sorting** refers to the sorting of an array of data that is in RAM (quick-, merge-, insertion-, selection-, radix-, bucket-, bubble-, heap-, sort algorithms we saw in the Data Struct. & Alg. Course)

• **Objective:** Minimize number of I/O accesses.

• External Sorting is part of the **Query Evaluation / Optimization subsystem**
  – Efficient Sorting algorithms can speed up query evaluation plans (e.g., during joins)!
Lecture Outline
External Sorting – Εξωτερική Ταξινόμηση

13.1) **When** does a DBMS sort Data.

13.2) A **Simple Two-Way Merge-Sort** (Απλή Εξωτερική Ταξινόμηση με Συγχώνευση)

13.3) **External Merge-Sort** (Εξωτερική Ταξινόμηση με Συγχώνευση)
   - Exclude 13.3.1: Minimizing the Number of Runs.

13.4.2) **Double Buffering** (Διπλή Προκαταχώρηση)

13.5) **Using B+Trees for Sorting**
When Does a DBMS Sort Data? (~30% of oper.)

- Data requested in sorted order
  - e.g., `SELECT * FROM Students ORDER BY gpa DESC;`
- Sorting is first step in bulk loading a B+ tree index.
  - i.e., `CREATE INDEX StuAge ON Students(age) USING BTREE;`
  - Recall how leaf nodes of the B+tree are ordered.
- Useful for eliminating duplicate copies in a collection of records.
  - `SELECT DISTINCT age FROM Students;`
  - i.e., to eliminate duplicates in a sorted list requires only the comparison of each element to its previous element so this yields a linear order elimination algorithm.
Two-Way External Merge-Sort

(Απλή Εξωτερική Ταξινόμηση με Συγχώνευση)

- Let us consider the simplest idea for external sorting
  - Assumption: Only 3 Buffer pages are available
  - Idea: Divide and Conquer (similarly to MergeSort, Quicksort)

- Idea Outline
  - Pass 0 (Sort Lists): For every page, read it, sort it, write it out
    - Only one buffer page is used!
    - Now we need to merge them hierarchically
  - Pass 1, 2, …, etc. (Merge Lists): see next page for merging concept
    - For this step we need three buffer pages!

**Diagram:**
- **Pass 0**
  - Main Memory Sort
  - Buffer
  - Input 1
  - Input 2
  - Output
  - Disk
  - Main memory

- **Passes 1, 2,…**
Merging Lists
(Συγχώνευση 2 Λιστών)

- **Merging Lists Outline (Phase 1,2,…)**
  1. Load the next sorted runs \( R1 \) and \( R2 \) into main memory buffers \( B1 \) and \( B2 \) a page-at-a-time (i.e., initially first page from each run) (see left figure)
     - Obviously \( R1 \geq B1 \) and \( R2 \geq B2 \) (a Run might be larger than a Buffer)
     - The rest pages will be loaded to main memory during subsequent steps.
  2. Initialize indices \( i, j \) to the head of each list (i.e., \( i=j=0 \))
     - If \( B1[i] \) was smallest item then \( i++ \) else \( j++ \) (see right figure)
     - If OUTPUT gets full, it is appended to the end of a file on DISK and cleared in RAM.
  4. Repeat the above until either index \( i \) or \( j \) reaches the end of its buffer.
     - At this point write the remaining records to OUTPUT, flush it to disk and finish.
  5. Repeat procedure from 1-4 until all runs have been traversed.
Two-Way External Merge Sort Example

(Pαράδειγμα Απλής Εξωτερικής Ταξινόμηση με Συγχώνευση)

Example Execution

Input file

PASS 0
1-page runs

PASS 1
2-page runs

PASS 2
4-page runs

PASS 3

PASS 4
Cost of Two-Way External Merge Sort

(Κόστος Απλής Εξωτερικής Ταξινόμηση με Συγχώνευση)

- Each pass we read + write each of N pages in file.

Number of passes:

\[ \floor{\log_2 N} + 1 \]

- e.g., for N=7, N=5 and N=4

\[ \floor{\log_2 7} + 1 = \floor{\log_{10} 7 \div \log_{10} 2} + 1 = \floor{2.8} + 1 = 4 \]

\[ \floor{\log_2 5} + 1 = \floor{2.3} + 1 = 4 \]

\[ \floor{\log_2 4} + 1 = \floor{2} + 1 = 3 \]

- Total (I/O) cost is:

\[ 2N \times (\# \text{ passes}) \]

- e.g., for N=7

\[ 2 \times 7 \times (\floor{\log_2 7} + 1) = 2 \times 7 \times 4 = 56 \]

- i.e., (read+write) * 7 pages * 4 passes

- That can be validated on the right figure

- Pass#0=2*7  Pass#1=2*7
- Pass#2=2*7  Pass$3=2*7
Two-Way External Merge-Sort

(Απλή Εξωτερική Ταξινόμηση με Συγχώνευση)

Two-Way External MergeSort Pseudocode

```plaintext
proc 2-way_extsort (file)
    // Given a file on disk, sorts it using three buffer pages
    // Produce runs that are one page long: Pass 0
    Read each page into memory, sort it, write it out.
    // Merge pairs of runs to produce longer runs until only
    // one run (containing all records of input file) is left
    While the number of runs at end of previous pass is > 1:
        // Pass i = 1, 2, ...
        While there are runs to be merged from previous pass:
            Choose next two runs (from previous pass).
            Read each run into an input buffer; page at a time.
            Merge the runs and write to the output buffer;
            force output buffer to disk one page at a time.
    endproc
```

Phase 0

We want to end up with 1 final run

Merging Lists
Internal Sorting Remarks
(Επισημάνσεις Αλγ. Ταξινόμησης Κύριας Μνήμης)

- External Sorting Algorithms utilize some **Internal Sorting** Algorithm to sort records in main memory.

- **Where Does In-Memory Sorting happens?**
  - **Dedicated Sorting Region** (e.g., IBM, Informix, Oracle)
  - **In general purpose Buffer Manager** (e.g., Microsoft and Sybase IQ for data warehousing)
  - **Elsewhere**, e.g., cache of recently executed stored procedures (e.g., Sybase ASE for OnLine Transaction Processing)

- **What In-Memory Algorithms do real DBMSes use?**
  - **Oracle** uses **InsertionSort**
  - **Microsoft** and **Sybase ASE** use **MergeSort**
  - **IBM** and **Sybase IQ** use **RadixSort** (quite similar to **BucketSort**)

- All systems utilize **Asynchronous (Ασύγχρονο) I/O** and **Prefetching (Προανάκτηση)** … we shall see the benefits of these techniques in the subsequent slides.
General External Merge Sort
(Γενικευμένη Εξωτερική Ταξινόμηση με Συγχώνευση)

- Let’s turn the 2-way Merge-sort Algorithm into a Practical Alg.
  - **Assumption:** B Buffer pages are available
  - **Idea:** Merge (B-1) pages in each step rather than only 2 (faster!)

- **Idea Outline**
  - **Pass 0 (Sort):** Sort the N pages using B buffer pages
    - Use B buffer pages for input
    - That generates \(N1 = \left\lceil \frac{N}{B} \right\rceil\) sorted runs e.g., \(N=8\) and \(B=4\) => \(N1 = \left\lceil \frac{8}{4} \right\rceil = 2\)
  - **Pass 1, 2, …, etc. (Merge):** Perform a (B-1)-way merge of runs
    - Use (B-1) buffer pages for input + 1 page for output
    - Number of passes will be reduced dramatically! (see slide 13)
Cost of External Merge Sort
(Kόστος Γενικευμένης Εξωτερική Ταξινόμηση με Συγχώνευση)

- Number of passes: \[ 1 + \left\lfloor \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rfloor \]
- I/O Cost = \[ 2N \times \text{(# of passes)} \]

Example: \( N=108 \) pages and \( B=5 \) Buffer pages

- #Passes = \[ 1 + \left\lfloor \log_4 \left\lceil \frac{108}{5} \right\rceil \right\rfloor = 4 \]
- IO Cost = Read+Write 108 pages for 4 passes => \( 2 \times 108 \times 4 = 864 \) IOs

  - Pass 0: \[ \left\lceil \frac{108}{5} \right\rceil = 22 \text{ sorted runs} \]
    of 5 pages each (last run is only 3 pages)
  
  - Pass 1: \[ \left\lceil \frac{22}{4} \right\rceil = 6 \text{ sorted runs} \]
    (5 runs x 20 pages each and last run is only 8 pages)

  - Pass 2: \[ \left\lceil \frac{6}{4} \right\rceil = 2 \text{ sorted runs} \]
    80 pages (4 x 20 pages) and 28 pages

  - Pass 3: Sorted file of 108 pages
### Number of Passes of External Sort

- External Merge Sort is quite efficient!
- With only **B=257 (~1MB) Buffer Pages** we can sort **N=1 Billion records** with four (4) passes … in practice B will be larger
- Two-Way Mergesort would require $\lceil \log_2 10^9 \rceil + 1 = 30 + 1$ passes!

<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>

*Results generated with formula: $1 + \left\lceil \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil$*
External Merge-Sort
(Εξωτερική Ταξινόμηση με Συγχώνευση)

External MergeSort Pseudocode

```plaintext
proc extsort (file)
// Given a file on disk, sorts it using three buffer pages
// Produce runs that are B pages long: Pass 0
Read B pages into memory, sort them, write out a run.
// Merge B – 1 runs at a time to produce longer runs until only
// one run (containing all records of input file) is left
While the number of runs at end of previous pass is > 1:
    // Pass i = 1, 2, ...
    While there are runs to be merged from previous pass:
        Choose next B – 1 runs (from previous pass).
        Read each run into an input buffer; page at a time.
        Merge the runs and write to the output buffer;
        force output buffer to disk one page at a time.
endproc
```

Phase 0
Double Buffering
(Διπλή Προκαταχώρηση)

- An I/O request takes time to complete. Only think about all the involved layers and delays (Disk Delays, Buffer Manager, etc).
- To reduce wait time of CPU for I/O request to complete, can prefetch (προανάκτηση) into `shadow block’ (μπλοκ αντίγραφο).
- Main Idea: When all tuples of \(INPUT_i\) have been consumed, the CPU can process \(INPUT_i’\) which is prefetched into main memory instead of waiting for \(INPUT_i\) to refill. … same idea applies to OUTPUT.

(B main memory buffers, k-way merge)
Using B+ Trees for Sorting
(Χρήση B+ Tree για Ταξινόμηση)

• **Scenario:** Table to be sorted has **B+ tree index** on sorting column(s).

• **Idea:** Can retrieve records in order by traversing leaf pages. *Is this a good idea?*

• Cases to consider:
  - B+ tree is **clustered**
    • *Always Good idea!*
  - B+ tree is **not clustered**
    • *Could be a very bad idea!*
Clustered B+ Tree Used for Sorting

(Ταξινόμηση με Ομαδοποιημένο Ευρετήριο)

- **Idea:** From root go to the left-most leaf, then retrieve all next leaf pages (Alternative 1)
- **Cost:** $\log_F N$, where $F$ is the branching factor.
- If **Alternative 2** is used? Additional cost of retrieving data records: however each page is only fetched once (and not not 3-4 rounds like external sorting)

*Always better than external sorting!*
Unclustered B+ Tree Used for Sorting
(Ταξινόμηση με Μη-Ομαδοποιημένο Ευρετήριο)

- **How about an Unclustered Index?** i.e.,
  Altern.(2) or Altern.(3) (note: Altern.(1) is always Clustered)

- **Cost:** In general, **one I/O per data record!**

- **This solution is really bad!**

![Diagram of Unclustered B+ Tree Used for Sorting](image-url)