Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#12: External Sorting (R&G, Ch13)

Last Class

- Static Hashing
- Extendible Hashing
- Linear Hashing
- Hashing vs. B-trees

Today's Class

- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for Sorting

Why Do We Need to Sort?

- **`SELECT`**...**`ORDER BY`**
- *Bulk loading* B+ tree index.
- *Duplicate elimination* (**`DISTINCT`**)
- **`SELECT`**...**`GROUP BY`**
- *Sort-merge* join algorithm.

Why Do We Need to Sort?

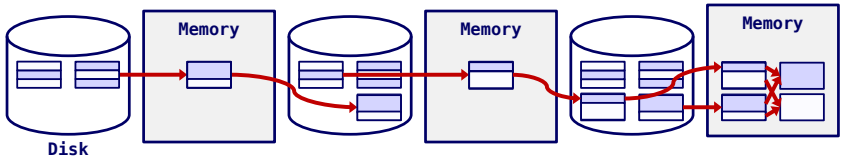
- What do we do if the data that we want to sort is larger than the amount of memory that is available to the DBMS?
- What if multiple queries are running at the same time and they all want to sort data?
- *Why not just use virtual memory?*

Overview

- Files are broken up into ***N*** pages.
- The DBMS has a finite number of ***B*** fixed-size buffers.
- *Let's start with a simple example…*

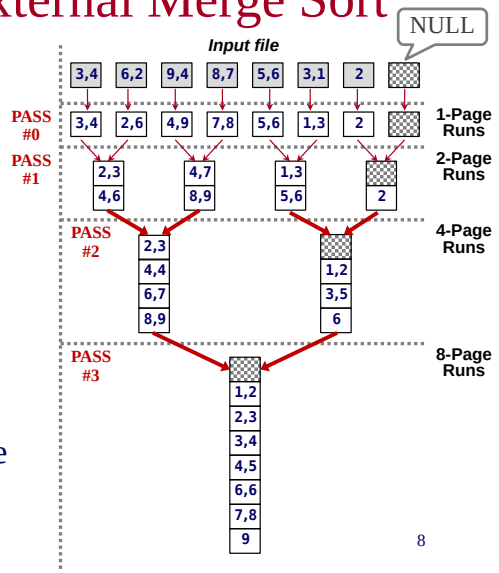
Two-way External Merge Sort

- **Pass 0**: Read a page, sort it, write it.
  - Only one buffer page is used
- **Pass 1,2,3,…**: requires 3 buffer pages
  - Merge pairs of **runs** into runs twice as long
  - Three buffer pages used.



Two-way External Merge Sort

- Each pass we read + write each page in file.
- # of passes
  = ⌈ $\log_2 N$ ⌉ + 1
- So total I/O cost is:
  = $2N \cdot$ (⌈ $\log_2 N$ ⌉ + 1)
- ***Divide and conquer*:**
  Sort subfiles and merge



Two-way External Merge Sort

- This only requires **three** buffer pages.
- Even if we have more buffer space available, this algorithm does not utilize it effectively.
- *Let's look at the general algorithm…*

General External Merge Sort

- ***B>3*** buffer pages.
- How to sort a file with ***N*** pages?

General External Merge Sort

- **Pass 0**: Use ***B*** buffer pages. Produce **⌈*N* / *B*⌉** sorted runs of size ***B***.
- **Pass 1,2,3,…**: Merge ***B-1*** runs.
- Number of passes: $1+\lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = **2N·(# of passes)**

Example

- Sort 108 page file with 5 buffer pages:
  - **Pass 0: ⌈108 / 5⌉** = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - **Pass 1: ⌈22 / 4⌉** = 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

$1+\lceil \log_{B-1} \lceil N / B \rceil \rceil = 1+\lceil \log_4 22 \rceil = 1+\lceil 2.229... \rceil$ → **4 passes**

Today's Class

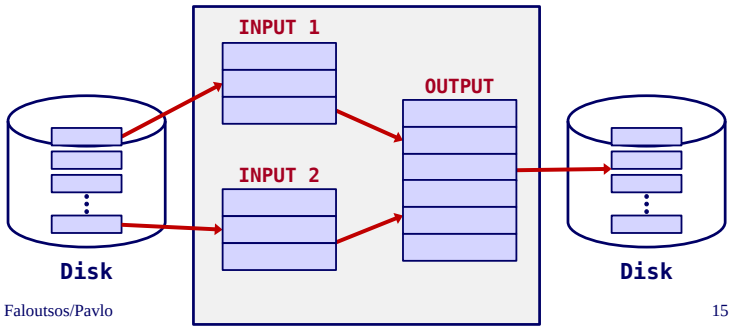
- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for Sorting

Blocked I/O & Double-buffering

- So far, we assumed random disk access.
- The cost changes if we consider that runs are written (and read) sequentially.
- What could we do to exploit it?
  - **Blocked I/O:** Exchange a few random reads for several sequential ones using bigger pages.
  - **Double-buffering:** Mask I/O delays with prefetching.

Blocked I/O

- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes



Blocked I/O

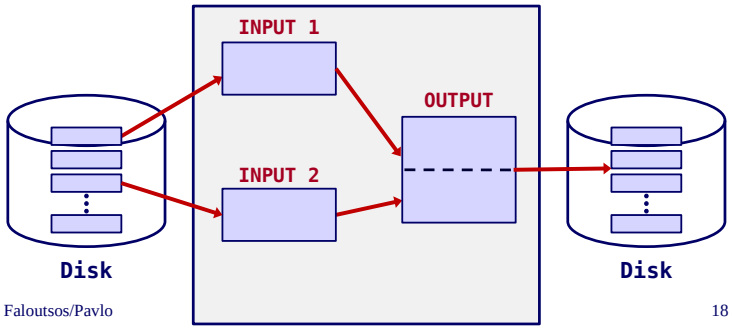
- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes
- Advantages?
  Fewer random disk accesses because some of them are sequential.
- Disadvantages?
  Less parallelization.

Blocked I/O & Double-buffering

- So far, we assumed random disk access
- Cost changes, if we consider that runs are written (and read) sequentially
- What could we do to exploit it?
  - **Blocked I/O:** Exchange a few random reads for several sequential ones using bigger pages.
  - **Double-buffering:** Mask I/O delays with prefetching.

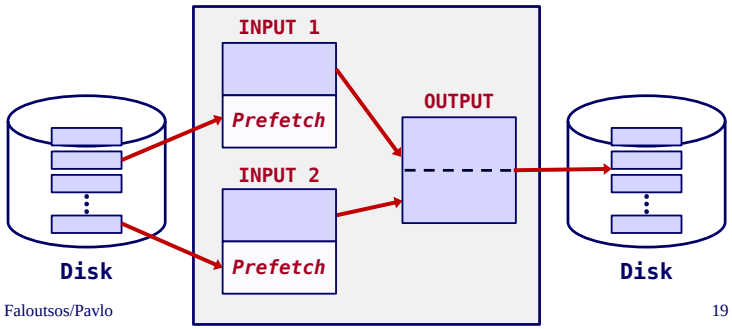
Double-buffering

- Normally, when, say 'INPUT1' is exhausted
  - We issue a "read" request and then we wait…



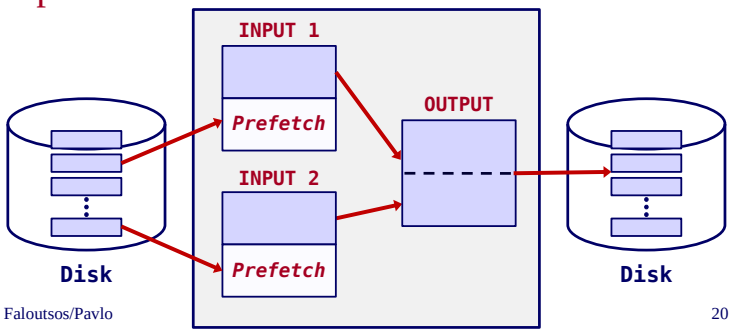
Double-buffering

- We can prefetch the next block into a spare buffer using a asynchronous thread.



Double-buffering

- This potentially requires more passes.
- But in practice, most files still sorted in 2-3 passes.



Today's Class

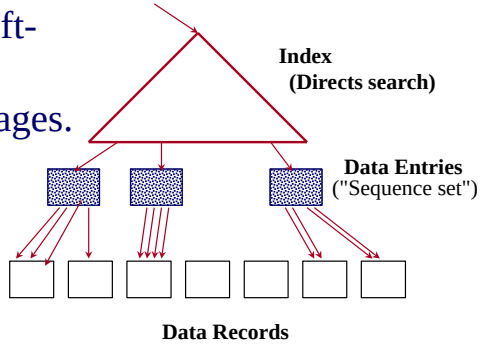
- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for Sorting

Using B+ Trees for Sorting

- 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 ***Good Idea!***
  - B+ tree is not clustered ***Could be Bad!***

Clustered B+Tree for Sorting

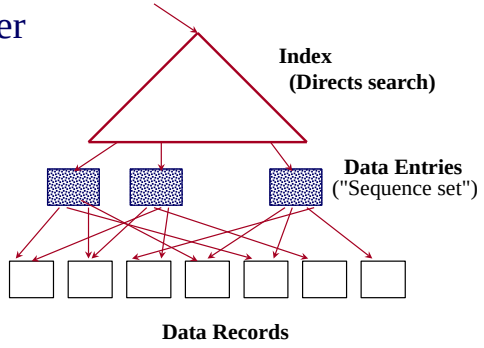
- Traverse to the left-most leaf, then retrieve all leaf pages.



Always better than external sorting!

Unclustered B+Tree for Sorting

- Chase each pointer to the page that contains the data.



In general, one I/O per data record!

Summary

- External sorting is important.
- External merge sort minimizes disk I/O:
  - **Pass 0:** Produces sorted ***runs*** of size ***B*** (# buffer pages).
  - **Later Passes**: Merge ***runs***.
- Clustered B+ tree is good for sorting; unclustered tree is usually bad.