


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

C. Faloutsos – A. Pavlo
 Lecture#12: External Sorting



Today's Class

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

Faloutsos/Pavlo CMU SCS 15-415/615 4



Why do we need to sort?

Faloutsos/Pavlo CMU SCS 15-415/615 5

CMU SCS

Why do we need to sort?

- **SELECT...ORDER BY**
 - e.g., find students in increasing *gpa* order
- *Bulk loading* B+ tree index.
- *Duplicate elimination* (**DISTINCT**)
- **SELECT...GROUP BY**
- *Sort-merge* join algorithm involves sorting.

Faloutsos/Pavlo CMU SCS 15-415/615 6

CMU SCS

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?*

Faloutsos/Pavlo CMU SCS 15-415/615 7

CMU SCS

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...*

Faloutsos/Pavlo CMU SCS 15-415/615 8

CMU SCS

Two-way 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.

Faloutsos/Pavlo 9

CMU SCS

Two-way External Merge Sort

- Each pass we read + write each page in file.

Faloutsos/Pavlo 10

CMU SCS

Two-way External Merge Sort

- Each pass we read + write each page in file.

Faloutsos/Pavlo 11

CMU SCS

Two-way External Merge Sort

- Each pass we read + write each page in file.

Faloutsos/Pavlo

CMU SCS

Two-way External Merge Sort

- Each pass we read + write each page in file.

Faloutsos/Pavlo

CMU SCS

Two-way External Merge Sort

- Each pass we read + write each page in file.
- N pages in the file => $\lceil \log_2 N \rceil + 1$
- So total cost is: $2N (\lceil \log_2 N \rceil + 1)$
- Divide and conquer:** sort subfiles and merge

Faloutsos/Pavlo

CMU SCS

Two-way External Merge Sort

- This algorithm 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...*

Faloutsos/Pavlo 15-415/615 15

CMU SCS

General External Merge Sort

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

The diagram shows two disk icons on the left and right, each containing several horizontal bars representing pages. In the center, a box labeled 'B Main memory buffers' contains several orange rectangular blocks of varying sizes, representing the buffers used to read and write pages during the sorting process.

Faloutsos/Pavlo 15-415/615 16

CMU SCS

General External Merge Sort

- **Pass 0:** Use B buffer pages. Produce $\lceil N / B \rceil$ sorted runs of B pages each.
- **Pass 1,2,3,...:** Merge $B-1$ runs.

The diagram shows two disk icons on the left and right. The left disk contains several horizontal bars representing sorted runs. Arrows point from these runs to a central box labeled 'B Main memory buffers'. Inside this box, there are orange blocks labeled 'INPUT 1', 'INPUT 2', and 'INPUT B-1'. Arrows from these input blocks point to an orange block labeled 'OUTPUT'. An arrow then points from the 'OUTPUT' block to the right disk, which contains several horizontal bars representing the merged sorted runs.

Faloutsos/Pavlo 15-415/615 17

CMU SCS

Sorting

- Create sorted runs of size B (how many?)
- Merge them (how?)

Faloutsos/Pavlo 15-415/615 18

CMU SCS

Sorting

- Create sorted runs of size B
- Merge first $B-1$ runs into a sorted run of $(B-1) \cdot B, \dots$

Faloutsos/Pavlo 15-415/615 19

CMU SCS

Sorting

- How many steps we need to do? i , where $B \cdot (B-1)^i > N$
- How many reads/writes per step? $N+N$

Faloutsos/Pavlo 15-415/615 20

CMU SCS

Cost of External Merge Sort

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- **Cost** = $2N \cdot (\# \text{ of passes})$

Faloutsos/Pavlo 15-415/615 21

CMU SCS

Example

- Sort 108 page file with 5 buffer 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

Formula check: $\lceil \log_4 22 \rceil = 3 \dots + 1 \rightarrow 4$ passes ✓

Faloutsos/Pavlo 15-415/615 22

CMU SCS

of Passes of External Sort

Cost = $2N \cdot (\# \text{ of passes})$

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Faloutsos/Pavlo 15-415/615 23

CMU SCS

Today's Class

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

Faloutsos/Pavlo CMU SCS 15-415/615 24

CMU SCS

Optimizations

- Which internal sort algorithm should we use for **Phase 0**?
- How do we prevent the DBMS from blocking when it needs input?

Faloutsos/Pavlo CMU SCS 15-415/615 25

CMU SCS

Internal Sort Algorithm

- **Quicksort** is a fast way to sort in memory.
- But we get **B** buffers, and produce one run of length **B** each time.
- Can we produce longer runs than that?

Faloutsos/Pavlo 15-415/615 26

CMU SCS

Heapsort

- Alternative sorting algorithm (a.k.a. "replacement selection")
- Produces runs of length $\sim 2 \cdot B$
- Clever, but **not** implemented, for subtle reasons: tricky memory management on variable length records

Faloutsos/Pavlo 15-415/615 27

CMU SCS

Reminder: Heapsort

pick smallest, write to output buffer:

Faloutsos/Pavlo 15-415/615 28

CMU SCS

Reminder: Heapsort

pick smallest, write to output buffer:

Faloutsos/Pavlo 15-415/615 29

CMU SCS

Reminder: Heapsort

get next key; put at top and 'sink' it

```
graph TD; 22[22] --> 14[14]; 22 --> 11[11]; 14 --> 15[15]; 14 --> 17[17]; 11 --> 18[18]; 11 --> 16[16];
```

Faloutsos/Pavlo 15-415/615 30

CMU SCS

Reminder: Heapsort

get next key; put at top and 'sink' it

```
graph TD; 11[11] --> 14[14]; 11 --> 22[22]; 14 --> 15[15]; 14 --> 17[17]; 22 --> 18[18]; 22 --> 16[16];
```

Faloutsos/Pavlo 15-415/615 31

CMU SCS

Reminder: Heapsort

get next key; put at top and 'sink' it

```
graph TD; 11[11] --> 14[14]; 11 --> 16[16]; 14 --> 15[15]; 14 --> 17[17]; 16 --> 18[18]; 16 --> 22[22];
```

Faloutsos/Pavlo 15-415/615 32

CMU SCS

Reminder: Heapsort

```

graph TD
    11[11] --- 14[14]
    11 --- 16[16]
    14 --- 15[15]
    14 --- 17[17]
    16 --- 18[18]
    16 --- 22[22]
            
```

When done, pick top (= smallest) and output it, if 'legal' (ie., ≥ 10 in our example)

This way, we can keep on reading new key values (beyond the B ones of quicksort)

Faloutsos/Pavlo 15-415/615 33

CMU SCS

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?

Faloutsos/Pavlo 15-415/615 34

CMU SCS

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 r.d.a for several sequential ones using bigger pages.
 - **Double-buffering:** mask I/O delays with prefetching.

Faloutsos/Pavlo 15-415/615 35

CMU SCS

Blocked I/O

- Normally, B buffers of size (say) 4K

Faloutsos/Pavlo 15-415/615 36

CMU SCS

Blocked I/O

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

Faloutsos/Pavlo 15-415/615 37

CMU SCS

Blocked I/O

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

Faloutsos/Pavlo 15-415/615 38

CMU SCS

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?

Faloutsos/Pavlo 15-415/615 39

CMU SCS

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?
Smaller fanout may cause more passes.

Faloutsos/Pavlo 15-415/615 40

CMU SCS

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 r.d.a for several sequential ones using bigger pages.
 - ➔ – **Double-buffering**: mask I/O delays with prefetching.

Faloutsos/Pavlo 15-415/615 41

CMU SCS

Double-buffering

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

Faloutsos/Pavlo 15-415/615 42

CMU SCS

Double-buffering

- We *prefetch* INPUT1' into "shadow block"
 - When INPUT1 is exhausted, we issue a "read",
 - BUT we proceed with INPUT1'

Faloutsos/Pavlo 15-415/615 43

CMU SCS

Double-buffering

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

Faloutsos/Pavlo 15-415/615 44

CMU SCS

Today's Class

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

Faloutsos/Pavlo CMU SCS 15-415/615 45

CMU SCS

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!*

Faloutsos/Pavlo 15-415/615 46

CMU SCS

Clustered B+ Tree for Sorting

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

Data Records

Always better than external sorting!

Faloutsos/Pavlo 15-415/615 47

CMU SCS

Unclustered B+ Tree for Sorting

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

In general, one I/O per data record!

Faloutsos/Pavlo 15-415/615 48

CMU SCS

External Sorting vs. Unclustered Index

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

N: # pages
p: # of records per page
B=1,000 and block size=32 for sorting
p=100 is the more realistic value.

Faloutsos/Pavlo 49

CMU SCS

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 very bad.

Faloutsos/Pavlo 15-415/615 50
