


CMU SCS

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

C. Faloutsos – A. Pavlo
Lecture#12: External Sorting (R&G, Ch13)




CMU SCS

Last Class

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

Faloutsos/Pavlo CMU SCS 15-415/615 2




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 3



CMU SCS

Why do we need to sort?

Faloutsos/Pavlo CMU SCS 15-415/615 4

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 5

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 6

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 7

CMU SCS

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.

Faloutsos/Pavlo 8

Two-way External Merge Sort

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

The diagram illustrates the merging process over four passes:

- PASS 0:** Input file with 1-page runs: [3,4], [6,2], [9,4], [8,7], [5,6], [3,1], [2].
- PASS 1:** 2-page runs: [2,3], [4,6], [4,7], [8,9], [1,3], [5,6], [2].
- PASS 2:** 4-page runs: [2,3], [4,4], [6,7], [8,9], [1,2], [3,5], [6].
- PASS 3:** 8-page runs: [1,2], [2,3], [3,4], [4,5], [6,6], [7,8], [9].

Faloutsos/Pavlo 9

Two-way External Merge Sort

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

The diagram illustrates the merging process over four passes:

- PASS 0:** Input file with 1-page runs: [3,4], [6,2], [9,4], [8,7], [5,6], [3,1], [2].
- PASS 1:** 2-page runs: [2,3], [4,6], [4,7], [8,9], [1,3], [5,6], [2].
- PASS 2:** 4-page runs: [2,3], [4,4], [6,7], [8,9], [1,2], [3,5], [6].
- PASS 3:** 8-page runs: [1,2], [2,3], [3,4], [4,5], [6,6], [7,8], [9].

Faloutsos/Pavlo 10

Two-way External Merge Sort

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

The diagram illustrates the merging process over four passes:

- PASS 0:** Input file with 1-page runs: [3,4], [6,2], [9,4], [8,7], [5,6], [3,1], [2].
- PASS 1:** 2-page runs: [2,3], [4,6], [4,7], [8,9], [1,3], [5,6], [2].
- PASS 2:** 4-page runs: [2,3], [4,4], [6,7], [8,9], [1,2], [3,5], [6].
- PASS 3:** 8-page runs: [1,2], [2,3], [3,4], [4,5], [6,6], [7,8], [9].

Faloutsos/Pavlo 11

Two-way External Merge Sort

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

The diagram illustrates the merging process over four passes:

- PASS 0:** Input file with 1-page runs: [3,4], [6,2], [9,4], [8,7], [5,6], [3,1], [2].
- PASS 1:** 2-page runs: [2,3], [4,6], [4,7], [8,9], [1,3], [5,6], [2].
- PASS 2:** 4-page runs: [2,3], [4,4], [6,7], [8,9], [1,2], [3,5], [6].
- PASS 3:** 8-page runs: [1,2], [2,3], [3,4], [4,5], [6,6], [7,8], [9].

Faloutsos/Pavlo 12

Two-way External Merge Sort

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

Faloutsos/Pavlo 13

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 14

General External Merge Sort

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

Faloutsos/Pavlo 15-415/615 15

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.

Faloutsos/Pavlo 15-415/615 16

CMU SCS

Sorting

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

Faloutsos/Pavlo 15-415/615 17

CMU SCS

Sorting

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

Faloutsos/Pavlo 15-415/615 18

CMU SCS

Sorting

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

Faloutsos/Pavlo 15-415/615 19

CMU SCS

Cost of External Merge Sort

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

$$\text{Cost} = 2 * N * (1 + \lceil \log_{B-1} [N/B] \rceil)$$

Faloutsos/Pavlo 15-415/615 20

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 21

CMU SCS

of Passes of External Sort

Cost = 2N · (# 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 22

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 23

CMU SCS

Optimizations

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

Faloutsos/Pavlo CMU SCS 15-415/615 24

CMU SCS

Internal Sort Algorithm DETAILS

- **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?
- *Longer Runs = Fewer Passes*

Faloutsos/Pavlo 15-415/615 25

CMU SCS

Heapsort DETAILS

- Alternative sorting algorithm (a.k.a. “replacement sort”) for Pass 0.
- 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 26

CMU SCS

Reminder: Heapsort DETAILS

pick smallest, write to output buffer:

Faloutsos/Pavlo 15-415/615 27

CMU SCS

Reminder: Heapsort DETAILS

pick smallest, write to output buffer:

Faloutsos/Pavlo 15-415/615 28

CMU SCS

Reminder: Heapsort

DETAILS

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 29

CMU SCS

Reminder: Heapsort

DETAILS

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 30

CMU SCS

Reminder: Heapsort

DETAILS

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 31

CMU SCS

Reminder: Heapsort

DETAILS

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)

```

    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

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 33

CMU SCS

Blocked I/O & Double-buffering

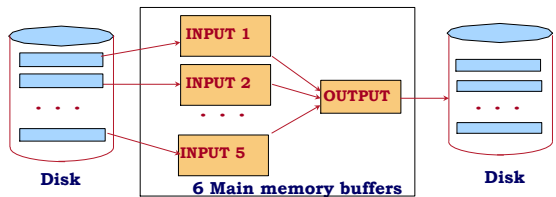
- So far, we assumed random disk access.
- The cost changes if we consider that runs are written (and read) sequentially.
- What could we do to exploit it?
 - ➔ – **Blocked I/O**: exchange a few r.d.a for several sequential ones using bigger pages.
 - **Double-buffering**: mask I/O delays with prefetching.

Faloutsos/Pavlo 15-415/615 34

CMU SCS

Blocked I/O

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

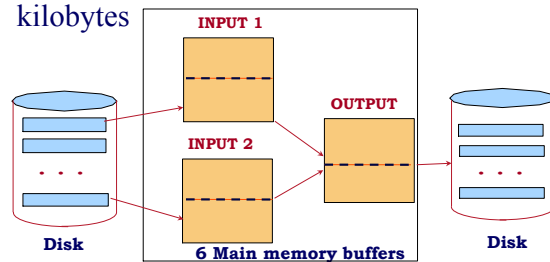


Faloutsos/Pavlo 15-415/615 35


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 36




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 37



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 38



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 39



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 40

CMU SCS

Double-buffering

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

Disk B Main memory buffers Disk

Faloutsos/Pavlo 15-415/615 41

CMU SCS

Double-buffering

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

Disk B Main memory buffers Disk

Faloutsos/Pavlo 15-415/615 42

CMU SCS

Double-buffering

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

Disk B Main memory buffers Disk

Faloutsos/Pavlo 15-415/615 43

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 44

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 45

CMU SCS

Clustered B+ Tree for Sorting

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

Always better than external sorting!

Faloutsos/Pavlo 15-415/615 46

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 47

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 48

CMU SCS

Sorting World Record

- Current Champions 2014 (tie) (TB/min)
 - TritonSort (UCSD)
 - 100 TB in 1,378 seconds (4.3 TB/min)
186 Amazon EC2 i2.8xlarge nodes
 - Apache Spark
 - 100 TB in 1,406 seconds (4.27 TB/min)
207 Amazon EC2 i2.8xlarge nodes
- More Info (\$/TB, etc)
 - <http://sortbenchmark.org>



Jim Gray

Faloutsos/Pavlo CMU SCS 15-415/615 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