EXTERNAL SORTING

CS 564- Spring 2018

ACKs: Dan Suciu, Jignesh Patel, AnHai Doan
WHAT IS THIS LECTURE ABOUT?

I/O aware algorithms for sorting

- External merge
  - a primitive for sorting

- External merge-sort
  - basic algorithm
  - optimizations
WHY SORTING?

• users often want the data sorted (ORDER BY)
• first step in bulk-loading a B+ tree
• used in duplicate elimination (why?)
• the sort-merge join algorithm (later in class) involves sorting as a first step
SORTING IN DATABASES

Why don’t the standard sorting algorithms work for a database system?

• merge sort
• quick sort
• heap sort

The data typically does not fit in memory!

E.g. how do we sort 1TB of data with 8GB of RAM?
EXTERNAL MERGE
EXTERNAL MERGE PROBLEM

Input: 2 sorted lists (with $M$ and $N$ pages)
Output: 1 merged sorted list (with $M+N$ pages)

Can we efficiently (in terms of I/O) merge the two lists using a buffer of size at least 3?

Yes, using only $2(M+N)$ I/Os!
EXTERNAL MERGE ALGORITHM

1, 5  8, 13  18, 20

3, 19  21, 22  24, 25

merged list

list 1

list 2

buffer  B = 3 frames

read

write
EXTERNAL MERGE ALGORITHM

**disk**

- list 1: 8, 13, 18, 20
- list 2: 21, 22, 24, 25

**buffer**

- B = 3 frames
  - 1, 5
  - 3, 19

**merged list**

- read 1 page from each list

---

read: 8, 13, 18, 20
write: 21, 22, 24, 25

CS 564 [Spring 2018] - Paris Koutris
EXTERNAL MERGE ALGORITHM

merge from the 2 pages until a new page is filled

disk

list 1

8, 13
18, 20

list 2

21, 22
24, 25

merged list

buffer

B = 3 frames

5
19
1, 3

read
write
EXTERNAL MERGE ALGORITHM

```
list 1
  8, 13
  18, 20

list 2
  21, 22
  24, 25

merged list
  1, 3
```

```
disk

buffer
  B = 3 frames
  5
  19
```

read

write

write the page to disk
EXTERNAL MERGE ALGORITHM

disk

list 1
8, 13
18, 20

list 2
21, 22
24, 25

merged list
1, 3

buffer
B = 3 frames

read

write

keep merging until one frame becomes empty
EXTERNAL MERGE ALGORITHM

list 1
- 18, 20

list 2
- 21, 22, 24, 25

merged list
- 1, 3

disk

read

buffer
- 8, 13
- 19
- 5

B = 3 frames

since 5 < 19, we know we should read from the first list!
EXTERNAL MERGE ALGORITHM

**List 1:**
- 18, 20

**List 2:**
- 21, 22
- 24, 25

**Buffer:**
- 13
- 19
- 5, 8

**Disk:**
- 1, 3

**B = 3 frames**

**Continue merging**
EXTERNAL MERGE ALGORITHM

list 1
- 18, 20

list 2
- 21, 22
- 24, 25

merged list
- 1, 3
- 5, 8

disk

buffer
- B = 3 frames

13
19

read
write

write to disk again
EXTERNAL MERGE ALGORITHM

- Disk
  - List 1
    - 18, 20
  - List 2
    - 21, 22
    - 24, 25
  - Merged list
    - 1, 3
    - 5, 8

- Buffer
  - B = 3 frames
    - 19
    - 13

Read

Write

and so on ...
EXTERNAL MERGE COST

We can merge 2 sorted lists of $M$ and $N$ pages using 3 buffer frames with

$I/O\ cost = 2 (M+N)$

When we have $B+1$ buffer pages, we can merge $B$ lists with the same I/O cost
EXTERNAL MERGE SORT
THE SORTING PROBLEM

- B available pages in buffer pool
- a relation R of size N pages (where N > B)

**SORTING**: output the same relation sorted on a given attribute
KEY IDEA

• split into chunks small enough to sort in memory (called runs)
• merge groups of runs using the external merge algorithm
• keep merging the resulting runs (each time is called a pass) until left with a single sorted file
**WARM UP: 2-WAY SORT**

```
buffer

B = 3 frames
N = 6 pages

disk

unsorted file

40, 3  8, 34  23, 12
2, 13  5, 17  25, 15

read

write

CS 564 [Spring 2018] - Paris Koutris 20```
WARM UP: 2-WAY SORT

Disk:
- 40, 3
- 8, 34
- 23, 12
- 2, 13
- 5, 17
- 25, 15

Buffer:
- B = 3 frames
- N = 6 pages

Split into chunks that fit in memory

Read

Write
**WARM UP: 2-WAY SORT**

- **disk**
  - 2, 13
  - 5, 17
  - 25, 15

- **buffer**
  - 40, 3
  - 8, 34
  - 23, 12

- **read**
- **write**

B = 3 frames
N = 6 pages

read each chunk in memory
WARM UP: 2-WAY SORT

\[ \text{buffer} \]

\[ \text{sort in memory} \]

\[ \text{disk} \]

\[ 2, 13 \]
\[ 5, 17 \]
\[ 25, 15 \]

\[ \text{read} \]

\[ 3, 8 \]
\[ 12, 23 \]
\[ 34, 40 \]

\[ \text{write} \]

\[ B = 3 \text{ frames} \]
\[ N = 6 \text{ pages} \]

CS 564 [Spring 2018] - Paris Koutris

23
WARM UP: 2-WAY SORT

Each sorted sub-file is called a **run**

**disk**

3, 8  12, 23  34, 40

2, 13  5, 17  25, 15

**buffer**

B = 3 frames
N = 6 pages

write back to disk
WARM UP: 2-WAY SORT

B = 3 frames
N = 6 pages

disk

3, 8  12, 23  34, 40

2, 5  13, 15  17, 25

buffer

read

write

now we have 2 runs!

same for the other chunk
WARM UP: 2-WAY SORT

**disk**

3, 8  12, 23  34, 40

2, 5  13, 15  17, 25

**buffer**

**final step**: use the external sort merge algorithm to merge the 2 runs

$B = 3$ frames  
$N = 6$ pages
CALCULATING THE I/O COST

In our example, $B = 3$ buffer pages, $N = 6$ pages

• Pass 0: creating the first runs
  – 1 read + 1 write for every page
  – total cost = $6 \times (1 + 1) = 12$ I/Os

• Pass 1: external merge sort
  – total cost = $2 \times (3 + 3) = 12$ I/Os

So 24 I/Os in total
I/O COST: SIMPLIFIED VERSION

Assume for now that we initially create \( N \) runs, each run consisting of a single page

- **pass 0**: \( N \) runs, each 1 page
- **pass 1**: merge into \( N/2 \) runs
- **pass 2**: merge into \( N/4 \) runs

- We need \( \lceil \log_2 N \rceil + 1 \) passes to sort the whole file
- Each pass needs \( 2N \) I/Os

\[
\text{total I/O cost} = 2N(\lceil \log_2 N \rceil + 1)
\]
CAN WE DO BETTER?

• The 2-way merge algorithm only uses 3 buffer pages
• But we have more available memory!

**Key idea**: use as much of the available memory as possible in every pass
• reducing the number of passes reduces I/O
EXTERNAL SORT: I/O COST

Suppose we have $B \geq 3$ buffer pages available

$2N([\log_2 N] + 1) \rightarrow 2N(\left\lceil \log_2 \frac{N}{B} \right\rceil + 1) \rightarrow 2N(\left\lceil \log_{B-1} \frac{N}{B} \right\rceil + 1)$

- initial runs of length 1
- 3-way merge
- increase the length of the initial runs to $B$
- merge $B-1$ runs at a time
## NUMBER OF PASSES

<table>
<thead>
<tr>
<th>$N$</th>
<th>$B=3$</th>
<th>$B=17$</th>
<th>$B=257$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10,000</td>
<td>13</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>20</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>10,000,000</td>
<td>23</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>26</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>30</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
Optimizing Merge Sort
REPLACEMENT SORT

- used as an alternative for the sorting in pass 0
- creates runs of *average* size $2B$ (instead of $B$)

**Algorithm**

- read $B-1$ pages in memory (keep as sorted heap)
- move smallest record (that is greater than the largest element in buffer) to output buffer
- read a new record $r$ and insert into the sorted heap
REPLACEMENT SORT: EXAMPLE

31, 12  10, 33  44, 55
18, 22  60, 24  3, 90

B = 3 frames
REPLACEMENT SORT: EXAMPLE

disk

44, 55
18, 22
60, 24
3, 90

buffer

31, 12
10, 33

read

write

B = 3 frames

load 2 pages, and create a heap file
REPLACEMENT SORT: EXAMPLE

B = 3 frames

start popping elements from the heap file, until page is full
REPLACEMENT SORT: EXAMPLE

- Disk:
  - 18, 22
  - 60, 24
  - 3, 90
  - 10, 12

- Buffer:
  - 31, 33
  - 44, 55

- Heap file:
  - Current max = 12

B = 3 frames

write to disk
load another page
REPLACEMENT SORT: EXAMPLE

- disk:
  - 18, 22
  - 60, 24
  - 3, 90
  - 10, 12

- buffer:
  - 44, 55
  - 31, 33

- heap file

B = 3 frames

pop more elements from heap

current max = 12

read

write

CS 564 [Spring 2018] - Paris Koutris
**REPLACEMENT SORT: EXAMPLE**

**disk**
- 60, 24
- 3, 90
- 10, 12
- 31, 33

**buffer**
- 18, 22
- 44, 55

**current max = 33**

**write to disk**
**load one more page**

**B = 3 frames**

**B = 3 frames**

**read**

**write**
REPLACEMENT SORT: EXAMPLE

**Current Max = 33**

**Now 18, 22 cannot be written to disk, since they are smaller than the current max**
REPLACEMENT SORT: EXAMPLE

disk

3, 90

10, 12  31, 33  44, 55

buffer

18, 22

60, 24

heap file

read

write

B = 3 frames

current max = 55

and so on ...
REPLACEMENT SORT: EXAMPLE

B = 3 frames

disk

3, 90

10, 12  31, 33
44, 55

buffer

18, 22
24
60

heap file

read

write

current max = 55

and so on ...

CS 564 [Spring 2018] - Paris Koutris
REPLACEMENT SORT: EXAMPLE

 disk

 3, 90

 10, 12

 44, 55

 10, 12

 31, 33

 60

 31, 33

 B = 3 frames

 buffer

 18, 22

 24

 18, 22

 current max = 60

 we stop here!

 read

 write

 heap file
I/O COST WITH REPLACEMENT SORT

Each initial run has length $\sim 2B$

$$I/O\ cost = 2N\left(\left\lfloor log_{B-1} \frac{N}{2B} \right\rfloor + 1\right)$$