CSE 132C
Database System Implementation

Arun Kumar

Topic 3: External Sorting

Chapter 13 of Cow Book

Slide ACKs: Jignesh Patel, Paris Koutris
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
Motivation for Sorting
Motivation for Sorting

❖ User’s SQL query has ORDER BY clause!
❖ First step of bulk loading of a B+ tree index
❖ Used in implementations of many relational ops: project, join, set ops, group by aggregate, etc. (next topic!)
Motivation for Sorting

❖ User’s SQL query has ORDER BY clause!
❖ First step of bulk loading of a B+ tree index
❖ Used in implementations of many relational ops: project, join, set ops, group by aggregate, etc. (next topic!)

Q: But sorting is well-known; why should a DBMS bother?
Motivation for Sorting

❖ User’s SQL query has ORDER BY clause!
❖ First step of bulk loading of a B+ tree index
❖ Used in implementations of many relational ops: project, join, set ops, group by aggregate, etc. (next topic!)

Q: But sorting is well-known; why should a DBMS bother?

Often, the file (relation) to be sorted will not fit in RAM!

“External” Sorting
External Sorting: Overview
Goal: Given relation $R$ with $N$ pages, $\textbf{SortKey} A$, $M$ buffer pages (often, $M << N$), sort $R$ on $A$ to get sorted $R'$.
External Sorting: Overview

❖ **Goal**: Given relation $R$ with $N$ pages, $\textbf{SortKey}$ $A$, $M$ buffer pages (often, $M << N$), sort $R$ on $A$ to get sorted $R'$

❖ **Idea**: Sorting algorithm should be disk page I/O-aware!
External Sorting: Overview

❖ **Goal**: Given relation $R$ with $N$ pages, **SortKey** $A$, $M$ buffer pages (often, $M << N$), sort $R$ on $A$ to get sorted $R'$

❖ **Idea**: Sorting algorithm should be disk page I/O-aware!

❖ **Desiderata**:
  - High efficiency, i.e., low I/O cost, even for very large $N$
  - Use sequential I/Os rather than random I/Os AMAP
  - Interleave I/O and comp. (DMA); reduce CPU cost too
External Sorting: Overview

- **Goal**: Given relation $R$ with $N$ pages, **SortKey** $A$, $M$ buffer pages (often, $M << N$), sort $R$ on $A$ to get sorted $R'$
- **Idea**: Sorting algorithm should be disk page I/O-aware!
- **Desiderata**:
  - High efficiency, i.e., low I/O cost, even for very large $N$
  - Use sequential I/Os rather than random I/Os AMAP
  - Interleave I/O and comp. (DMA); reduce CPU cost too

**NB**: I/O-aware sorting is also a key part of the implementation of MapReduce/Hadoop!
Warm-up: 2-way External Merge Sort
Warm-up: 2-way External Merge Sort

**Idea**: Make Merge Sort I/O-aware!
Warm-up: 2-way External Merge Sort

**Idea**: Make Merge Sort I/O-aware!

1. **Sort phase**: Read each page into buffer memory; do “internal” sort (use any popular fast sorting algorithm, e.g., quicksort); write it back to disk (a sorted “run”)
Warm-up: 2-way External Merge Sort

Idea: Make Merge Sort I/O-aware!

1. **Sort phase**: Read each page into buffer memory; do “internal” sort (use any popular fast sorting algorithm, e.g., quicksort); write it back to disk (a sorted “run”)

2. **Merge phase**: Read 2 runs, merge them on the fly, write out a new $double$-$length$ run; recurse till whole file is a run!
Warm-up: 2-way External Merge Sort

Idea: Make Merge Sort I/O-aware!

1. **Sort phase**: Read each page into buffer memory; do “internal” sort (use any popular fast sorting algorithm, e.g., quicksort); write it back to disk (a sorted “run”)

2. **Merge phase**: Read 2 runs, merge them on the fly, write out a new *double-length* run; recurse till whole file is a run!

**NB**: Sort phase is 1-pass; merge phase is often multi-pass!
Warm-up: 2-way External Merge Sort
Warm-up: 2-way External Merge Sort

Input file
Warm-up: 2-way External Merge Sort

Input file
PASS 0
1-page runs
Warm-up: 2-way External Merge Sort

Input file
PASS 0
1-page runs
PASS 1
2-page runs

Warm-up: 2-way External Merge Sort

Input file
PASS 0
1-page runs
PASS 1
2-page runs
Warm-up: 2-way External Merge Sort

Input file
PASS 0
1-page runs
PASS 1
2-page runs
Warm-up: 2-way External Merge Sort

Input file
PASS 0
1-page runs
PASS 1
2-page runs
PASS 2
4-page runs
Warm-up: 2-way External Merge Sort

Input file

PASS 0
3,4
6,2
9,4
8,7
5,6
3,1
2

1-page runs

PASS 1
2,3
4,6
4,7
8,9
1,3
5,6
2

2-page runs

PASS 2
2,3
4,4
6,7
8,9

4-page runs

PASS 3
1,2
2,3
3,4
4,5
6,6
7,8
9

8-page runs
Warm-up: 2-way External Merge Sort

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: \([\log_2(N)]\)

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: $[\log_2(N)]$

Each pass does 1 read and 1 write of whole file: $2N$ page I/Os per pass

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: $\lceil \log_2(N) \rceil$

Each pass does 1 read and 1 write of whole file:
2N page I/Os per pass

I/O cost of 2-way EMS:

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: \(\lceil \log_2(N) \rceil\)

Each pass does 1 read and 1 write of whole file: 2N page I/Os per pass

I/O cost of 2-way EMS:
\[2N \left(1 + \lceil \log_2(N) \rceil \right)\]

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: $[\log_2(N)]$

Each pass does 1 read and 1 write of whole file:
2N page I/Os per pass

I/O cost of 2-way EMS:
$$2N \left(1 + \lfloor \log_2(N) \rfloor \right)$$

N=7 pages

Whole file is sorted!
Warm-up: 2-way External Merge Sort

Number of passes:
Sort phase: 1
Merge phase: $\lceil \log_2(N) \rceil$

Each pass does 1 read and 1 write of whole file: 2N page I/Os per pass

I/O cost of 2-way EMS: $2N(1 + \lceil \log_2(N) \rceil)$

N=7 pages

=2*7*4=56

Whole file is sorted!
Warm-up: 2-way External Merge Sort

- Number of passes:
- Sort phase: 1
- Merge phase: $\lceil \log_2(N) \rceil$

Each pass does 1 read and 1 write of whole file: 2N page I/Os per pass

I/O cost of 2-way EMS: $2N \left(1 + \lceil \log_2(N) \rceil \right)$

N=7 pages

$=2*7*4=56$

Q: How to reduce this cost further?

Whole file is sorted!
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
Multi-way EMS: Motivation
Q: How many buffer pages does 2-way EMS use?
**Q:** How many buffer pages does 2-way EMS use?

Sort phase: 1 or 2 (1 each for R and W / 1 for in-place sort)

Merge phase: 3 (1 for each run input; 1 for merged output)

So, 2-way EMS uses only 3 buffer pages!
Multi-way EMS: Motivation

**Q:** How many buffer pages does 2-way EMS use?
Sort phase: 1 or 2 (1 each for R and W / 1 for in-place sort)
Merge phase: 3 (1 for each run input; 1 for merged output)
So, 2-way EMS uses only 3 buffer pages!

**Idea:** Why not exploit more buffer pages (say, B >> 3)?
Multi-way EMS: Motivation

**Q:** How many buffer pages does 2-way EMS use?
Sort phase: 1 or 2 (1 each for R and W / 1 for in-place sort)
Merge phase: 3 (1 for each run input; 1 for merged output)
So, 2-way EMS uses only 3 buffer pages!

**Idea:** Why not exploit more buffer pages (say, \( B \gg 3 \))?

Sort phase:
Read \( B \) pages at a time (not just 1 at a time)!
Write out \( \left\lfloor \frac{N}{B} \right\rfloor \) sorted runs of length \( B \) each (not just 1)
But I/O cost of sort phase is still the same!
Multi-way EMS: B-way Merge Phase
Multi-way EMS: B-way Merge Phase

**Idea:** In 2-way EMS, we merge 2 sorted runs at a time; in multi-way EMS, we merge B-1 sorted runs at a time!
Multi-way EMS: B-way Merge Phase

Idea: In 2-way EMS, we merge 2 sorted runs at a time; in multi-way EMS, we merge B-1 sorted runs at a time!

B-1 way merge; total # buffer pages used: B
Multi-way EMS: B-way Merge Phase

Idea: In 2-way EMS, we merge 2 sorted runs at a time; in multi-way EMS, we merge B-1 sorted runs at a time!

B-1 way merge; total # buffer pages used: B

# passes for Merge Phase reduces to: \[ \log_{B-1}(\lceil N/B \rceil) \]
Multi-way EMS I/O Cost
Multi-way EMS I/O Cost

Overall, \# passes = 1 + \left\lfloor \log_{B-1}(\left\lfloor N/B \right\rfloor) \right\rfloor
Multi-way EMS I/O Cost

Overall, \# passes = \(1 + \left\lfloor \log_{B-1}\left(\left\lfloor \frac{N}{B} \right\rfloor \right) \right\rfloor\)

I/O cost per pass = \(2N\)
Multi-way EMS I/O Cost

Overall, # passes = $1 + \left\lfloor \log_{B-1}(\lceil N/B \rceil) \right\rfloor$

I/O cost per pass = $2N$

Total I/O cost of EMS = $2N(1 + \left\lfloor \log_{B-1}(\lceil N/B \rceil) \right\rfloor)$
Multi-way EMS I/O Cost

Overall, # passes = $1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil$

I/O cost per pass = $2N$

Total I/O cost of EMS = $2N \times (1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil)$

Example: File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS B=1000
Multi-way EMS I/O Cost

Overall, # passes = 1 + \[ \log_{B-1}(\lceil \frac{N}{B} \rceil) \]

I/O cost per pass = 2N

Total I/O cost of EMS = 2N \(1 + \log_{B-1}(\lceil \frac{N}{B} \rceil)\)

**Example**: File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS B=1000

Number of pages N = 100M * 0.5KB / 8KB = 6.25M
Multi-way EMS I/O Cost

Overall, # passes = \(1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil\)

I/O cost per pass = \(2N\)

Total I/O cost of EMS = \(2N(1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil)\)

**Example:** File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS B=1000

Number of pages \(N = 100M \times 0.5KB / 8KB = 6.25M\)

Total I/O cost of EMS = \(2 \times 6.25M \times (1 + \lceil \log_{999}(6250) \rceil)\)
Multi-way EMS I/O Cost

Overall, \# passes = \( 1 + \left\lfloor \log_{B-1} \left( \left\lceil \frac{N}{B} \right\rceil \right) \right\rfloor \)

I/O cost per pass = \( 2N \)

Total I/O cost of EMS = \( 2N \left( 1 + \left\lfloor \log_{B-1} \left( \left\lceil \frac{N}{B} \right\rceil \right) \right\rfloor \) \)

**Example:** File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS \( B=1000 \)

Number of pages \( N = 100M \times 0.5KB / 8KB = 6.25M \)

Total I/O cost of EMS = \( 2 \times 6.25M \times (1 + \left\lfloor \log_{999}(6250) \right\rfloor) \)
Multi-way EMS I/O Cost

Overall, # passes = $1 + \lceil log_{B-1}(\lfloor N/B \rfloor) \rceil$

I/O cost per pass = $2N$

Total I/O cost of EMS = $2N \cdot (1 + \lceil log_{B-1}(\lfloor N/B \rfloor) \rceil)$

**Example**: File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS B=1000

Number of pages $N = 100M \cdot 0.5KB / 8KB = 6.25M$

Total I/O cost of EMS = $2 \times 6.25M \times (1 + \lceil log_{999}(6250) \rceil)$

*Only need the ceil!*
Multi-way EMS I/O Cost

Overall, # passes = \( 1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil \)

I/O cost per pass = \( 2N \)

Total I/O cost of EMS = \( 2N(1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil) \)

Example: File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS \( B=1000 \)

Number of pages \( N = 100M \times 0.5KB / 8KB = 6.25M \)

Total I/O cost of EMS = \( 2 \times 6.25M \times (1 + \lceil \log_{999}(6250) \rceil) \)

= \( 2 \times 6.25M \times (1 + 2) \) \( \text{Only need the ceil!} \)

= 37.5M
Multi-way EMS I/O Cost
Multi-way EMS I/O Cost

Total number of passes = $1 + \left\lceil \log_{B-1}(\left\lfloor N/B \right\rfloor) \right\rceil$
Multi-way EMS I/O Cost

Total number of passes = \( 1 + \left\lceil \log_{B-1} \left( \left\lfloor \frac{N}{B} \right\rfloor \right) \right\rceil \)

<table>
<thead>
<tr>
<th>N</th>
<th>Naive 2-way EMS</th>
<th>B=1K</th>
<th>B=10K</th>
<th>B=100K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>21</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10M</td>
<td>25</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100M</td>
<td>28</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1B</td>
<td>31</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Multi-way EMS I/O Cost

Total number of passes = $1 + \left\lfloor \log_{B-1} \left( \left\lfloor \frac{N}{B} \right\rfloor \right) \right\rfloor$

<table>
<thead>
<tr>
<th>N</th>
<th>Naive 2-way EMS</th>
<th>B=1K</th>
<th>B=10K</th>
<th>B=100K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>21</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10M</td>
<td>25</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100M</td>
<td>28</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1B</td>
<td>31</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

With 8KB page, 8TB!
Multi-way EMS I/O Cost

Total number of passes = \(1 + \left[ \log_{B-1}(\ceil{N/B}) \right]\)

<table>
<thead>
<tr>
<th>N</th>
<th>Naive 2-way EMS</th>
<th>B=1K</th>
<th>B=10K</th>
<th>B=100K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>21</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10M</td>
<td>25</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100M</td>
<td>28</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1B</td>
<td>31</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

With 8KB page, 800MB

With 8KB page, 8TB!
# Multi-way EMS I/O Cost

Total number of passes = \(1 + \left\lfloor \log_{B-1}(\left\lceil N/B \right\rceil) \right\rfloor\)

<table>
<thead>
<tr>
<th>(N)</th>
<th>Naive 2-way EMS</th>
<th>(B=1\text{K})</th>
<th>(B=10\text{K})</th>
<th>(B=100\text{K})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>21</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10M</td>
<td>25</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100M</td>
<td>28</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1B</td>
<td>31</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

With 8KB page, 800MB

With 8KB page, 8TB!

Only 2 passes to sort up to ~79.9TB!

(2 is the lower bound for EMS!)
Multi-way EMS: Improvements
Multi-way EMS: Improvements

- While already efficient, some key algorithmic+systems-oriented improvements have been made to multi-way EMS to reduce *overall runtime* (not just counting I/O cost)
Multi-way EMS: Improvements

❖ While already efficient, some key algorithmic+systems-oriented improvements have been made to multi-way EMS to reduce overall runtime (not just counting I/O cost)
❖ Three prominent improvements:
  1. Replacement sort (aka heap sort) as internal sort
  2. “Blocked” I/O
  3. Double Buffering
Improvement 1: Replacement Sort
Improvement 1: Replacement Sort

- In standard EMS, quick sort used during Sort Phase
- Produces \([N/B]\) runs of length B pages each
Improvement 1: Replacement Sort

- In standard EMS, quick sort used during Sort Phase
- Produces \([N/B]\) runs of length B pages each
- Replacement sort is an alternative for Sort Phase
- Produces runs of *average* length 2B pages each
- So, number of runs reduced on average to \([N/2B]\)
Improvement 1: Replacement Sort

- In standard EMS, quick sort used during Sort Phase
- Produces $\lceil N/B \rceil$ runs of length B pages each
- Replacement sort is an alternative for Sort Phase
- Produces runs of average length 2B pages each
- So, number of runs reduced on average to $\lceil N/2B \rceil$
- Maintains a sorted heap in B-2 pages; 1 page for reading; 1 for sorted output
Improvement 1: Replacement Sort

- In standard EMS, quick sort used during Sort Phase
- Produces \( \lceil \frac{N}{B} \rceil \) runs of length B pages each
- Replacement sort is an alternative for Sort Phase
- Produces runs of average length 2B pages each
- So, number of runs reduced on average to \( \lceil \frac{N}{2B} \rceil \)
- Maintains a sorted heap in B-2 pages; 1 page for reading; 1 for sorted output
- Slightly higher CPU cost; but signif. lower I/O cost
Improvement 1: Replacement Sort

- In standard EMS, quick sort used during Sort Phase
  - Produces $\left\lfloor \frac{N}{B} \right\rfloor$ runs of length B pages each
- Replacement sort is an alternative for Sort Phase
  - Produces runs of average length 2B pages each
  - So, number of runs reduced on average to $\left\lfloor \frac{N}{2B} \right\rfloor$
  - Maintains a sorted heap in B-2 pages; 1 page for reading; 1 for sorted output
  - Slightly higher CPU cost; but signif. lower I/O cost

(We are skipping the details of this algorithm)
Improvement 1: Replacement Sort

❖ In standard EMS, quick sort used during Sort Phase
❖ Produces $\lfloor N/B \rfloor$ runs of length B pages each
❖ Replacement sort is an alternative for Sort Phase
❖ Produces runs of average length 2B pages each
❖ So, number of runs reduced on average to $\lfloor N/2B \rfloor$
❖ Maintains a sorted heap in B-2 pages; 1 page for reading; 1 for sorted output
❖ Slightly higher CPU cost; but signif. lower I/O cost

(We are skipping the details of this algorithm)

New total I/O cost = $2N(1 + \lceil \log_{B-1}(\lfloor N/2B \rfloor) \rceil)$
Improvement 2: “Blocked” I/O
Improvement 2: “Blocked” I/O

- Merge Phase did not recognize distinction between sequential I/O and random I/O!
- Time difference not reflected in counting I/O cost
Improvement 2: “Blocked” I/O

❖ Merge Phase did *not* recognize distinction between sequential I/O and random I/O!
❖ Time difference *not* reflected in counting I/O cost
❖ **Idea**: Read a “block” of b pages of each run at a time!
❖ So, only $\lceil B/b \rceil - 1$ runs can be merged at a time
❖ b controls trade-off of # passes vs time-per-pass
Improvement 2: “Blocked” I/O

❖ Merge Phase did not recognize distinction between sequential I/O and random I/O!
❖ Time difference not reflected in counting I/O cost
❖ **Idea**: Read a “block” of $b$ pages of each run at a time!
❖ So, only $\lfloor B/b \rfloor - 1$ runs can be merged at a time
❖ $b$ controls trade-off of # passes vs time-per-pass

“Fan-in” of Merge Phase = $F = \lfloor B/b \rfloor - 1$
Improvement 2: “Blocked” I/O

- Merge Phase did *not* recognize distinction between sequential I/O and random I/O!
- Time difference *not* reflected in counting I/O cost
- **Idea**: Read a “block” of $b$ pages of each run at a time!
- So, only $\lfloor B/b \rfloor - 1$ runs can be merged at a time
- $b$ controls trade-off of # passes vs time-per-pass

“Fan-in” of Merge Phase = $F = \lfloor B/b \rfloor - 1$

New total I/O cost = $2N(1 + \lceil \log_F(N') \rceil)$
Improvement 2: “Blocked” I/O

- Merge Phase did not recognize distinction between sequential I/O and random I/O!
- Time difference not reflected in counting I/O cost
- **Idea**: Read a “block” of $b$ pages of each run at a time!
- So, only $\lfloor B/b \rfloor - 1$ runs can be merged at a time
- $b$ controls trade-off of # passes vs time-per-pass

  “Fan-in” of Merge Phase = $F = \lfloor B/b \rfloor - 1$

  New total I/O cost = $2N(1 + \lfloor \log_F(N') \rfloor)$

  $N' = \lfloor N/B \rfloor$ or $N' = \lfloor N/2B \rfloor$
Improvement 3: Double Buffering
Improvement 3: Double Buffering

- Most machines have DMA; enables I/O-CPU parallelism
- Trivially feasible to exploit DMA in the Sort Phase
- But in the Merge Phase, CPU blocked by I/O for runs
Improvement 3: Double Buffering

- Most machines have DMA; enables I/O-CPU parallelism
- Trivially feasible to exploit DMA in the Sort Phase
- But in the Merge Phase, CPU blocked by I/O for runs

Idea: Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!
Improvement 3: Double Buffering

❖ Most machines have DMA; enables I/O-CPU parallelism
❖ Trivially feasible to exploit DMA in the Sort Phase
❖ But in the Merge Phase, CPU blocked by I/O for runs
❖ **Idea:** Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!
❖ So, only \( \lfloor B/2b \rfloor - 1 \) runs can be merged at a time
Improvement 3: Double Buffering

❖ Most machines have DMA; enables I/O-CPU parallelism
❖ Trivially feasible to exploit DMA in the Sort Phase
❖ But in the Merge Phase, CPU blocked by I/O for runs
❖ **Idea**: Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!
❖ So, only \( \lceil B/2b \rceil - 1 \) runs can be merged at a time

New fan-in of Merge Phase = \( F = \lfloor B/2b \rfloor - 1 \)
Improvement 3: Double Buffering

- Most machines have DMA; enables I/O-CPU parallelism
- Trivially feasible to exploit DMA in the Sort Phase
- But in the Merge Phase, CPU blocked by I/O for runs

**Idea**: Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!

- So, only \([B/2b] - 1\) runs can be merged at a time

New fan-in of Merge Phase = \(F = \lfloor B/2b \rfloor - 1\)

New total I/O cost = \(2N(1 + \lfloor log_F(N') \rfloor)\)
**Improvement 3: Double Buffering**

- Most machines have DMA; enables I/O-CPU parallelism
- Trivially feasible to exploit DMA in the Sort Phase
- But in the Merge Phase, CPU blocked by I/O for runs
- **Idea**: Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!
- So, only \( \lceil B/2b \rceil - 1 \) runs can be merged at a time

New fan-in of Merge Phase = \( F = \lceil B/2b \rceil - 1 \)

New total I/O cost = \( 2N(1 + \lceil \log_F(N') \rceil) \)

\( N' = \lceil N/B \rceil \) or \( N' = \lceil N/2B \rceil \)
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
External Sorting: Outline

❖ Overview and Warm-up
❖ Multi-way External Merge Sort (EMS)
❖ EMS and B+ trees
Using B+ Tree for EMS
Using B+ Tree for EMS

❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
❖ Data entries of the B+ tree are already in sorted order!
Using B+ Tree for EMS

❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
❖ Data entries of the B+ tree are already in sorted order!

Q: *Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?*
Using B+ Tree for EMS

- Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
- Data entries of the B+ tree are already in sorted order!

**Q: Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?**

It depends!
Using B+ Tree for EMS

❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
❖ Data entries of the B+ tree are already in sorted order!

Q: Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?

It depends!

On whether the index is clustered or not!
Using B+ Tree for EMS

❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
❖ Data entries of the B+ tree are already in sorted order!

Q: Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?

It depends!

On whether the index is clustered or not!

Good idea!
Using B+ Tree for EMS

❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
❖ Data entries of the B+ tree are already in sorted order!

Q: Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?

It depends!

On whether the index is clustered or not!

Good idea! Might be really bad!
Using Clustered B+ Tree for EMS
Using Clustered B+ Tree for EMS

❖ Go down the tree to reach left-most leaf
❖ Scan leaf pages (data entries) left to right
Using Clustered B+ Tree for EMS

❖ Go down the tree to reach left-most leaf
❖ Scan leaf pages (data entries) left to right
❖ If AltRecord, done! O/W, retrieve data pages pointed to by successive data entries
Using Clustered B+ Tree for EMS

- Go down the tree to reach left-most leaf
- Scan leaf pages (data entries) left to right
- If AltRecord, done! O/W, retrieve data pages pointed to by successive data entries

- I/O cost if AltRecord: height + # leaf pages
- I/O cost otherwise: height + # leaf pages + # data pages
Using Clustered B+ Tree for EMS

❖ Go down the tree to reach left-most leaf
❖ Scan leaf pages (data entries) left to right
❖ If AltRecord, done! O/W, retrieve data pages pointed to by successive data entries
❖ I/O cost if AltRecord: height + # leaf pages
❖ I/O cost otherwise: height + # leaf pages + # data pages
❖ Either way, I/O cost often << from-scratch EMS!
Using Unclustered B+ Tree for EMS
Using Unclustered B+ Tree for EMS

- Unclustered means not AltRecord! Why?
- Same procedure as for clustered B+ tree
Using Unclustered B+ Tree for EMS

❖ Unclustered means not AltRecord! Why?
❖ Same procedure as for clustered B+ tree
❖ Same I/O “cost” as for clustered tree with AltRID/AltRIDlist but many back-to-back *random* I/Os; thrashing!
Using Unclustered B+ Tree for EMS

- Unclustered means not AltRecord! Why?
- Same procedure as for clustered B+ tree
- Same I/O “cost” as for clustered tree with AltRID/AltRIDlist but many back-to-back random I/Os; thrashing!
- Usually, much slower than from-scratch EMS!
Using Unclustered B+ Tree for EMS

❖ Unclustered means not AltRecord! Why?
❖ Same procedure as for clustered B+ tree
❖ Same I/O “cost” as for clustered tree with AltRID/AltRIDlist but many back-to-back random I/Os; thrashing!
❖ Usually, much slower than from-scratch EMS!

Q: But when is this faster than from-scratch EMS?
# External Sorting as Competitive Sport!

The geekiest “sport” in the world: [sortbenchmark.org](http://sortbenchmark.org)

<table>
<thead>
<tr>
<th></th>
<th>Daytona</th>
<th>Indy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gray</td>
<td>2016, 44.8 TB/min</td>
<td>2016, 60.7 TB/min</td>
</tr>
<tr>
<td></td>
<td><strong>Tencent Sort</strong></td>
<td><strong>Tencent Sort</strong></td>
</tr>
<tr>
<td></td>
<td>100 TB in 134 Seconds</td>
<td>100 TB in 98.8 Seconds</td>
</tr>
<tr>
<td></td>
<td>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,</td>
<td>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,</td>
</tr>
<tr>
<td></td>
<td>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,</td>
<td>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,</td>
</tr>
<tr>
<td></td>
<td>100Gb Mellanox ConnectX4-EN)</td>
<td>100Gb Mellanox ConnectX4-EN)</td>
</tr>
<tr>
<td></td>
<td>Jie Jiang, Lixiong Zheng, Junfeng Pu,</td>
<td>Jie Jiang, Lixiong Zheng, Junfeng Pu,</td>
</tr>
<tr>
<td></td>
<td>Xiong Cheng, Chongqing Zhao</td>
<td>Xiong Cheng, Chongqing Zhao</td>
</tr>
<tr>
<td></td>
<td>Tencent Corporation</td>
<td>Tencent Corporation</td>
</tr>
<tr>
<td></td>
<td>Mark R. Nutter, Jeremy D. Schaub</td>
<td>Mark R. Nutter, Jeremy D. Schaub</td>
</tr>
<tr>
<td>Cloud</td>
<td>2016, $1.44 / TB</td>
<td>2016, $1.44 / TB</td>
</tr>
<tr>
<td></td>
<td><strong>NADSort</strong></td>
<td><strong>NADSort</strong></td>
</tr>
<tr>
<td></td>
<td>100 TB for $144</td>
<td>100 TB for $144</td>
</tr>
<tr>
<td></td>
<td>394 Alibaba Cloud ECS ecs.n1.large nodes x</td>
<td>394 Alibaba Cloud ECS ecs.n1.large nodes x</td>
</tr>
<tr>
<td></td>
<td>(Haswell E5-2680 v3, 8 GB memory,</td>
<td>(Haswell E5-2680 v3, 8 GB memory,</td>
</tr>
<tr>
<td></td>
<td>40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk)</td>
<td>40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk)</td>
</tr>
<tr>
<td></td>
<td>Qian Wang, Rong Gu, Yihua Huang</td>
<td>Qian Wang, Rong Gu, Yihua Huang</td>
</tr>
<tr>
<td></td>
<td>Nanjing University</td>
<td>Nanjing University</td>
</tr>
<tr>
<td></td>
<td>Reynold Xin</td>
<td>Reynold Xin</td>
</tr>
<tr>
<td></td>
<td>Databricks Inc.</td>
<td>Databricks Inc.</td>
</tr>
<tr>
<td></td>
<td>Wei Wu, Jun Song, Junluan Xia</td>
<td>Wei Wu, Jun Song, Junluan Xia</td>
</tr>
<tr>
<td></td>
<td>Alibaba Group Inc.</td>
<td>Alibaba Group Inc.</td>
</tr>
<tr>
<td>Minute</td>
<td>2016, 37 TB</td>
<td>2016, 55 TB</td>
</tr>
<tr>
<td></td>
<td><strong>Tencent Sort</strong></td>
<td><strong>Tencent Sort</strong></td>
</tr>
<tr>
<td></td>
<td>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,</td>
<td>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,</td>
</tr>
<tr>
<td></td>
<td>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,</td>
<td>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,</td>
</tr>
<tr>
<td></td>
<td>100Gb Mellanox ConnectX4-EN)</td>
<td>100Gb Mellanox ConnectX4-EN)</td>
</tr>
<tr>
<td></td>
<td>Jie Jiang, Lixiong Zheng, Junfeng Pu,</td>
<td>Jie Jiang, Lixiong Zheng, Junfeng Pu,</td>
</tr>
<tr>
<td></td>
<td>Xiong Cheng, Chongqing Zhao</td>
<td>Xiong Cheng, Chongqing Zhao</td>
</tr>
<tr>
<td></td>
<td>Tencent Corporation</td>
<td>Tencent Corporation</td>
</tr>
<tr>
<td></td>
<td>Mark R. Nutter, Jeremy D. Schaub</td>
<td>Mark R. Nutter, Jeremy D. Schaub</td>
</tr>
</tbody>
</table>

2013, 168,242 Joules 2013, 168,242 Joules
Review Questions: Indexing and EMS

1. What is the difference between a B tree and B+ tree?
2. Between the insert and delete operations of a B+ tree index, when/where is redistribution of entries not preferred?
3. Why is a hash index not useful for a range search?
4. Briefly explain 1 pro and 1 con of the extendible hash index over the static hash index.
5. Is it possible to somehow combine at least some of the benefits of B+ tree and hash indexes? :)
6. Why bother optimizing EMS when its time complexity is already at the lower bound of O(N log(N))? 
7. Which EMS optimization(s) can actually raise the total I/O cost?
8. Given a relation of size 100GB, 8GB buffer memory, and 8KB page size, what is the I/O cost (in GB) for sorting this relation?