CompSci 516
Data Intensive Computing Systems

Lecture 7
External Sorting

Instructor: Sudeepa Roy
Announcements

• **Project proposal informal email due today**
  – September 21 (Wednesday)
  – A few lines on the project + group members

• **Homework 2:**
  – due on Oct 12 (the same day as the midterm)
  – finish early -- should be ongoing!
  – remember to turn off instances

• **Midterm**
  – practice problems will be posted periodically
Reading Material

• (Complete Index from Lecture 5-6)

• [RG]
  – External sorting: Chapter 13

• [GUW]
  – Chapter 14

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
External Sorting
Why Sort?

• A classic problem in computer science
• Data requested in sorted order
  – e.g., find students in increasing gpa order
• Sorting is first step in bulk loading B+ tree index
• Sorting useful for eliminating duplicate copies in a collection of records
• Sort-merge join algorithm involves sorting
• Problem: sort 1Gb of data with 1Mb of RAM
  – need to minimize the cost of disk access
A simple 2-way sort

• Not too practical, but useful to learn basic concepts for external sorting
• Utilizes only 3 pages of main memory
• Several sorted sub-files are generated at intermediate steps
• Each sorted sub-file is called a run
  – each run can contain multiple pages
2-Way Sort: Requires 3 Buffers

• Suppose $N = 2^k$ pages in the file
• Pass 0: Read a page, sort it, write it.
  – repeat for all $2^k$ pages
  – only one buffer page is used
• Pass 1:
  – Read two pages, sort (merge) them using one output page, write them to disk
  – repeat $2^{k-1}$ times
  – three buffer pages used
• Pass 2, 3, 4, ..... continue
Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N pages in the file
- \( \Rightarrow \) the number of passes
  \( = \left\lfloor \log_2 N \right\rfloor + 1 \)
- So total cost is:
  \[ 2N\left\lfloor \log_2 N \right\rfloor + 1 \]
- Idea: Divide and conquer: sort subfiles and merge
General External Merge Sort

- Suppose we have more than 3 buffer pages.
- How can we utilize them?

- To sort a file with N pages using B buffer pages:
  - Pass 0: use B buffer pages:
    - Produce $\lceil N/B \rceil$ sorted runs of B pages each.
  - Pass 1, 2, ..., etc.: merge B-1 runs to one output page
    - keep writing to disk once the output page is full
Cost of External Merge Sort

- Number of passes: \(1 + \lceil \log_{B-1}[N/B] \rceil\)
- Cost = \(2N \times (# \text{ of passes})\) – why 2 times?
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: sorting 5 pages at a time
    - \([108/5] = 22\) sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: 4-way merge
    - \([22/4] = 6\) sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 4-way merge
    - (but 2-way for the last two runs)
      - \([6/4] = 2\) sorted runs, 80 pages and 28 pages
  - Pass 3: 2-way merge (only 2 runs remaining)
    - Sorted file of 108 pages
# Number of Passes of External Sort

High B is good, although CPU cost increases

<table>
<thead>
<tr>
<th>N</th>
<th>B=3</th>
<th>B=5</th>
<th>B=9</th>
<th>B=17</th>
<th>B=129</th>
<th>B=257</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10,000</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100,000</td>
<td>17</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,000,000</td>
<td>20</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10,000,000</td>
<td>23</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>26</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>30</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
I/O for External Merge Sort

• If 10 buffer pages
  – either merge 9 runs at a time with one output buffer
  – or 8 runs with two output buffers

• If #page I/O is the metric
  – goal is minimize the #passes
  – each page is read and written in each pass

• If we decide to read a block of b pages sequentially
  – Suggests we should make each buffer (input/output) be a block of pages
  – But this will reduce fan-out during merge passes
    • i.e. not as many runs can be merged again any more
  – In practice, most files still sorted in 2-3 passes
Double Buffering

• To reduce CPU wait time for I/O request to complete, can *prefetch* into `shadow block`.

```
Disk

INPUT 1
INPUT 1'
INPUT 2
INPUT 2'
INPUT k
INPUT k'

OUTPUT
OUTPUT'

b
block size

B main memory buffers, k-way merge

Disk
```
Using B+ Trees for Sorting

• Scenario: Table to be sorted has B+ tree index on sorting column(s)
• Idea: Can retrieve data entries (then 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 a very bad idea!
Clustered B+ Tree Used for Sorting

- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)

- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once

Always better than external sorting!
Unclustered B+ Tree Used for Sorting

• Alternative (2) for data entries; each data entry contains rid of a data record
• In general, one I/O per data record!
Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!

- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted runs of size $B$ (# buffer pages)
  - Later passes: merge runs
  - # of runs merged at a time depends on $B$, and block size.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller # runs merged.
  - In practice, # of runs rarely more than 2 or 3