External Sorting

CS634
Lecture 10, Mar 2, 2016

Slides based on "Database Management Systems" 3rd ed, Ramakrishnan and Gehrke

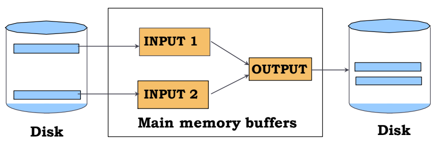
Why is Data Sorting Important?

- 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*
  - Needed for set operations, DISTINCT operator
- *Sort-merge join* algorithm involves sorting

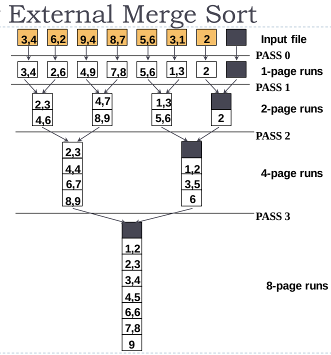
- Problem: sort 1Gb of data with 1MB of RAM, or 10MB
  - Sort is given a memory budget, can use temp disk as needed
  - Focus is minimizing I/O, not computation as in internal sorting

2-Way Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it
  - only one buffer page is used
- Pass 2, 3, …, etc.:
  - three buffer pages used



Two-Way External Merge Sort



Two-Way External Merge Sort

- Each pass we read + write each page in file.
- Number of pages N in the file determines number of passes
  Ex: N = 7, round up to power-of-two 8 = 2^3, #passes = 4 (last slide)
  Here 3 = $\log_2 8$ = ceiling($\log_2 7$), so 4 = ceiling($\log_2 N$) + 1
- Total number of passes is, using ceiling notation:

$$\lceil \log_2 N \rceil + 1$$

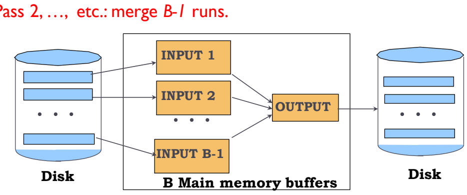
- Total cost is: write & read all N pages for each pass:

$$2N(\lceil \log_2 N \rceil + 1)$$

General External Merge Sort

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 2, …, etc.: merge *B-1* runs.



Cost of External Merge Sort, as on pg. 427, with yellow over over-simplistic conclusion: see next slide

- Example: with 5 buffer pages, sort 108 page file:
  - 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: ceiling(6/4) = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Merge 2 runs to produce sorted file of 108 pages

  Note 22 rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, each with one input buffer.

  3 = ceiling($\log_4$ 22) where 4 = B-1 and 22 = ceiling(N/B)

  plus the initial pass, so 4 passes in all.
- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = 2N * (# of passes) = 2*108*4 i/os
- This cost assumes the data is read from an input file and written to another output file, and this i/o is counted

Cost of External Merge Sort

- Example: with 5 buffer pages, sort 108 page file:
  - Pass 0: ceiling(108/4) = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: ceiling(22/4) = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: ceiling(6/4) = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Merge 2 runs into sorted file of 108 pages

  Note 22 rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, each with one input buffer.

  3 = ceiling($\log_4$ 22) where 4 = B-1 and 22 = ceiling(N/B)

  plus the initial pass, so 4 passes in all.
- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- But the passes are not always all the same size: look at writes and reads over whole run (including writing the output of the sort in the last pass)
  - [Read N], write N, read N, write N, read N, write N, read N, [write N]
  - The bracketed amounts depend on whether or not the data is read from a file at the start and written to a file at the end. Often data is pipelined in and/or pipelined out, saving significantly
- That's 6N, 7N, or 8N i/os, not always the 8N as given in the book's formula
- Cost = N * (# of read/writes of N) = 2N(#passes - 1) up to 2N(#passes)

Cost of External Merge Sort, bigger file

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = 2N * (# of passes)
- Example: with 5 buffer pages, sort 250 page file, including reading the input data from a file and writing the output data to another file.
  - Pass 0: ceiling(250/5) = 50 sorted runs of 5 pages each
  - Pass 1: ceiling(50/4) = 13 sorted runs of 20 pages each (last run is only 10 pages)
  - Pass 2: ceiling(13/4) = 4 sorted runs, 80 pages and 10 pages
  - Pass 3: Sorted file of 250 pages

  Note 50 again rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, plus the initial pass, so 4 passes again

  Cost = 2*250*4 i/os

  But 50 is getting up in the vicinity of 64, where we start needing another pass

Number of Passes of External Sort

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

Example of a Blocked I/O Sort

Example: N=1M blocks, B=5000 blocks memory for sort

Use 32 blocks in a big buffer, so have 5000/32 = 156 big buffers

File is 1M/32 = 31250 big blocks

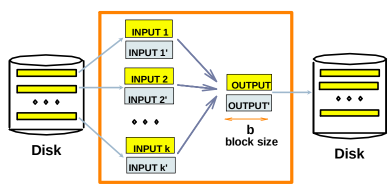
- Pass 0: sort using 156 big buffers to first runs: get ceiling(31250/156) = 201 runs
- Pass 1: merge using 155 big buffers to 2 runs
- Pass 2: merge 2 runs to final result

See 3 passes here, vs. 2 using "optimized" sort, pg. 431

- Cost = 2N*3 = 6N, vs. 4N using ordinary blocks
- But I/O is 4ms vs. (5/32)ms, so 6*(5/32)=1 vs. 4*4 = 16, a win.

Prefetching to speed up reading

- To reduce wait time for I/O request to complete, can *prefetch* into `*shadow block*'
  - Potentially, more passes; in practice, most files *still* sorted in 2-3 passes



B main memory buffers, k-way merge

Prefetching, tuning i/o

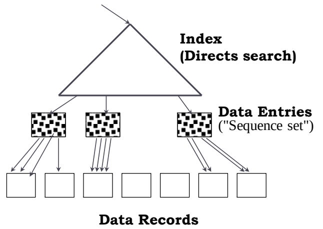
- Note this is a general algorithm, not just for sorting
- Can be used for table scans too
- Database have I/O related parameters
- **Oracle:**
- DB_FILE_MULTIBLOCK_READ_COUNT
- Says how many blocks to read at once in a table scan

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 a very bad idea!***

(Already existent) Clustered B+ Tree Used for Sorting

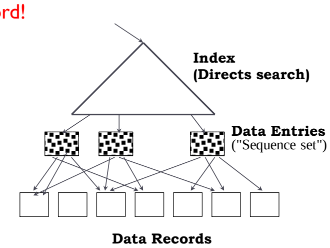
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used, additional cost of retrieving data records: each page fetched just once



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!



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

- ***p*: # of records per page (*p=100 is the more realistic value)***
- ***B=1,000 and block size=32 for sorting***
- ***Assumes the blocks are never found in the buffer pool***

Sorting Records: Benchmarks

- Parallel sorting benchmarks/competitions exist in practice
- Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: 3.5 ***seconds*** (circa1997)
    - 12-CPU SGI machine, 96 disks, 2GB of RAM
  - 2001: .48 sec. at UW (most recent I could find)
  - Oracle on dbs2: 3 min. using default settings, 24MB for PGA.
- Newer benchmarks:
  - Minute Sort: How many TB can you sort in 1 minute?
    2015: 7.7TB, using general purpose sort code on a system with 3,134 nodes each with 2 Xeon cores and 96GB memory) and … at Alibaba Group Inc.
  - Cloud Sort: How much in USD to sort 100 TB using a public cloud
    2015: $451 on 330 Amazon EC2 r3.4xlarge nodes, by profs at UCSD.

Oracle on dbs2: 3 min to sort 1M records

- In roughly 100MB of data (actually 78MB)
- Suppose Oracle allots 1MB for this sort
- Then 100MB/1MB = 100 runs in pass 0
- 1MB/8KB = 128 pages of buffer (B=128)
- So pass 1 merges 100 runs into final sorted output
- The DB reads/writes the 100MB twice, then the output is saved in the filesystem (faster, ignore for now)
- 200MB/8KB = 25K i/o, at about 100 i/o/s
- 25Ki/os/(100 i/os/s) = 250 s = 4 min
- Works out, so probably right # passes.

Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting! Oracle: separate memory area
- 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 passes rarely more than 2 or 3, for properly managed database and decent sized memory.

Summary, cont.

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs
- The best sorts are wildly fast:
  - Despite 40+ years of research, we're still improving!
- Clustered B+ tree is good for avoiding sorting; unclustered tree is usually useless.