CS W186 Introduction to Database Systems
Spring 2020 Josh Hug, Michael Ball

DIS 5

1 General External Merge Sort

Assumptions – N pages to sort, B buffer pages in memory
Pass 0 – Use B buffer pages. Produce $\frac{N}{B}$ sorted runs of B pages each.
Further passes – Merge $B-1$ runs.
Last pass – Produces 1 run of N pages.
Total I/O cost – $2N(\#\text{ of passes}) = 2N(1+\lceil \log_{B-1}\lceil \frac{N}{B} \rceil \rceil)$

(a) You have 4 buffer pages and your file has a total of 108 pages of records to sort. How many passes would it take to sort the file?

Pass 0 - ceil(108/4) = 27 sorted runs of 4 pages each
Pass 1 - ceil(27/3) = 9 sorted runs of 12 pages each
Pass 2 - ceil(9/3) = 3 sorted runs of 36 pages each
Pass 3 - Sorted file (1 run)
Total = 4 passes

(b) How many runs would each pass produce?

Pass 0 - 27 sorted runs (of 4 pages each)
Pass 1 - 9 sorted runs (of 12 pages each)
Pass 2 - 3 sorted runs (of 36 pages each)
Pass 3 - 1 sorted run (of 108 pages)

(c) What is the total cost for this sort process in terms of I/O?

4 passes * 2 (read + write per pass) * 108 (pages in the file) = 864 I/Os

(d) If the pages were already sorted individually, how many passes would it take to sort the file and how many IOs would it be instead?

Sorted pages does not change IO cost! Since Pass 0 is still going to produce ceil(N/B) sorted runs of B pages each, you would still require 4 passes and 864 IOs.

(e) If we wanted to sort N pages with B buffer pages in at most p total passes, write an expression relating the minimum buffer pages B needed with N and p. What do you notice about B when $p = 1$?

Since we want (# of passes after pass 0) $\leq p-1$, we set the equation $\log_{B-1}\frac{N}{B} \leq p-1$ (we remove the ceilings since we want to minimize B). Rearranging results in $B(B-1)^{p-1} \geq N$. If $p = 1$, this means that $B \geq N$ which, conceptually, means that if we want to sort N pages in 1 pass, all of them must fit into memory at the same time.

2 Hashing

Given – N pages to hash, B buffer pages in memory
Initial partitioning – Read in N pages and hash into $B-1$ partitions. Write out

$$\sum_{i=1}^{B-1} (\#\text{ of pages in partition i})$$

Recursive partitioning - For each individual partition, recursively partition if its size $s > B$.
Building in-memory hash table - Once a partition's size $s \leq B$, read in s pages, build in-memory hash table, and write out s pages.

(a) What are some use-cases in which hashing is preferred over sorting?

Removing duplicates, when partition phase can be omitted or shortened. Operations that require only data rendezvous (matching data must be together) and no order requirements - such as GROUP BY without ORDER BY.

(b) Suppose we have B buffer pages and can process $B(B-1)$ pages of data with External Hashing in two passes. For this case, fill in the blanks with the appropriate # of pages.
_____ input buffer(s)
_____ partitions after pass 0
_____ pages per partition

__1__ input buffer(s)
__B-1__ partitions after pass 0
__B__ pages per partition

(c) If you are processing exactly $B(B-1)$ pages of data with external hashing, is it likely that you'll have to perform recursive external hashing? Why or why not?

Yes. To avoid additional recursive external hashing, you would have to have an absolutely perfect hash function that evenly distributes records into the B - 1 partitions. This is almost impossible in practice. We should expect that some partitions may have more than B pages after partition hashing.

(d) We want to hash $N = 100$ pages using $B = 10$ buffer pages. Suppose in the initial partitioning pass, the pages are unevenly hashed into partitions of 10, 20, 20, and 50 pages. Assuming uniform hash functions are used for every partitioning pass after this pass, what is the total I/O cost for External Hashing?

634 I/Os. (I/Os are in bold) **100** pages are read and hashed into partitions of size 10, 20, 20, and 50. Summing these results in **100** pages written.

The partition of size 10 can fit into memory ($10 \leq (B = 11)$) so we can build the in-memory hash table for it. This involves reading in **10** pages, building the hash table, and writing **10** pages.

Since the partitions of sizes 20 and 50 cannot fit into memory, they must be recursively partitioned. For one of the partitions of size 20, **20** pages are read, hashed into 9 partitions of

size 3 ($\lceil \frac{20}{B-1} \rceil = \lceil \frac{20}{9} \rceil = 3$), and thus **27** pages are written (9 partitions x 3 pages/partition). Since each of these partitions of size 3 can now fit into memory, we read in a total of **27** pages, build the in-memory hash tables for each partition, and write out **27** pages. The same process after the initial partition is repeated for the other partition of size 20 which results in another $20+27+27+27 =$ **101** I/Os.

For the partition of size 50, **50** pages are read, hashed into 9 partitions of size 6 ($\lceil \frac{50}{B-1} \rceil = \lceil \frac{50}{9} \rceil = 6$), and thus **54** pages are written (9 partitions x 6 pages/partition). Since each of these partitions of size 6 can now fit into memory, we read in a total of **54** pages, build the in-memory hash tables for each partition, and write out **54** pages.

Summing all I/Os results in a total of **634** I/Os.

Here's a visualization of the hashing process. Squares represent partitions with its size, red arrows represent reads, blue arrows represent writes, a green square indicates that a partition can fit into memory and thus the in-memory hash table for it can be built, and circles represent building hash tables. Adding all reads and writes results in **634** I/Os.