2-Way Sort

Problem

- You have some large number (e.g., 3072) pages of data to sort
- You only have a small number (e.g., 3) pages to do it
- How do you do this?

Idea 1: Sort/Merge

Phase 1:

- Load 3 pages of data
- Sort everything
- Flush out this new sorted run of size 3 to disk
- Repeat until all data touched once

Phase 2

- Pick 2 sorted runs of size 3 and merge them together
  - Requires 2 pages from the 2 sorted runs
  - Requires 1 output page
  - As soon as an input page is empty, read in the next
  - As soon as an output page is full, flush it to disk
  - Repeat until all sorted runs of size 3 are merged into sorted runs of size 6

Phases 3 to 11 (or, in general, until done)

- As phase 2, but keep multiplying the sorted run size by 2

Cost Analysis:

- Phase 1: 3072 x 2 IOs (one read/write per page of data)
- Phase 2-11: 3072 x 2 IOs (one read/write per page of data)

In general:

- Phase 1 creates runs of size 3
• Phase 2 creates runs of size $3 \cdot 2^{\text{phase-1}}$

▼ Last phase is when $3 \cdot 2^{\text{phase-1}} \geq \#\text{pages}$
  • One sorted run of the full length of the data
  ▼ Equivalently:
    • $2^{\text{phase-1}} \geq \#\text{pages}/3$
    • phase-1 $\geq \log_2(\#\text{pages}/3)$
    • phases $\geq 1 + \log_2(\#\text{pages}/3)$
    • ceil(1 + log_2( #pages/3 )) phases required
    • Total: $\#\text{pages} \cdot 2 \cdot (1 + \log_2( \#\text{pages} / N ))$ IOs

▼ Idea 2: N-Way Sort/Merge
  • What if we have more than 3 pages (say we have N pages)?
  ▼ Phase 1:
    • Load N pages of data instead
  ▼ Phases 2 and onwards:
    • Simultaneously merge N-1 sorted runs
    • (optionally use some of the space to buffer reads/writes)
  ▼ Cost Analysis
    • Base cost per phase is still $\#\text{pages} \times 2$ IOs each
    • Now, last phase is at $N \cdot (N-1)^{\text{phase-1}} > \#\text{pages}$
    • So: ceil( 1 + log_(N-1)( #pages / N ) ) phases required

▼ Idea 3: Longer Initial Sorted Runs
  ▼ Using only N memory, can we create sorted runs longer than N?
    • Obviously, I wouldn’t ask if the answer was no.
  ▼ Idea: Flush data out a little at a time
    • Load N pages of data, sort in-memory
    • Flush the first page out to disk
    • Now you have a free page!
• Read in another page of unsorted data
• Sort the result in memory
• Repeat?

▼ Problem: What if you get a lower value than something you already flushed out?

• Keep track of the highest value flushed out to disk in the current sorted run.
• Don't flush out records below this value
• Instead, set them aside for the next sorted run
• Eventually you won't be able to flush any new records out... at this point, you end the current sorted run and start the next one

▼ Cost Analysis:

• On average, you have a 50% chance of getting a record lower than your highest flushed value
• Initial sorted runs will be ~2x as long, saving you 2/N phases

▼ Bonus

• What happens if the input is "already" sorted?
• ... or mostly sorted?

▼ Aggregation

▼ Overview

• Data is Big - Users often want summary statistics
• How do we compute these summary statistics efficiently?

▼ Fold

▼ An "iterator-style" operation with 2 parts

• A Default Value (e.g., 0)
• A Merge Current Value and Record Value operation (e.g., current + record)
\(\Box\) **COUNT**
- Default: 0
- Merge: current + 1

\(\Box\) **SUM**
- Default: 0
- Merge: current + record

\(\Box\) **MAX (resp, MIN)**
- Default: -\(\infty\)
- Merge: Max(current, record)

\(\Box\) **AVERAGE**
- Actually a combination of COUNT and SUM:
  - \(\text{SUM}(X) / \text{COUNT}(\ast)\)
- Can express as a fold over a tuple of values:
  - Default: \(<\text{count}: 0, \text{sum}: 0>\)
  - Merge: \(<\text{current.count} + 1, \text{current.sum} + \text{record}>\)
- Need a "finalize" step:
  - Finalize: current.sum / current.count

\(\Box\) **MEDIAN**
- Default: \(\emptyset\)
- Merge: current \(\bigcup\) record
- Finalize: Find the median

\(\Box\) **What gives?**
- Median is a "holistic" aggregate
- "Algebraic" aggregates have a constant-size intermediate result
- Holistic aggregates need all of the data (e.g., in sorted order)
Group-By Aggregation

What if you want multiple aggregate values?

- SELECT A, SUM(B) FROM R
  - Creates one row for each A, with a sum of all of the B values from rows with that A.
  - How do we implement this?

Idea 1: In-Memory Hash Table

- Scan records in any order
- For each record, check to see if the hash table contains the group by attribute(s) value(s)
  - If not, create a new entry in the hash table with the default group value
  - Incorporate the new record's aggregate value

Idea 2: Pre-Sort the Data

- Problem w/ Idea 1: What if you run out of memory
- Use the external sort algorithm above by the group-by attributes

Benefit: you know that all elements of a single group will be adjacent to one another:

- If you iterate over the sorted list of elements, as soon as the group by attributes change, you know you're done with that group
- ... so you only ever need to keep one "current value" in memory at a time
- Pro: You can start emitting intermediate results before you're done with everything
- Con: Log(N) full passes over the data

Idea 3: Pre-Hash the Data
• Do one pass through the data to create hash buckets that will fit in memory

▼ Like sorting, but you only need one pass through the data

• ... unless you guess wrong about the number of buckets to create