Disk Sorting and Disk Scheduling

CS157B

Chris Pollett

Jan 31, 2005.

Outline

- I/O Model of Computation
- Sorting in Secondary Storage
- Multiway Merge Sort
- Organizing Data by Cylinder
- Using Multiple Disks
- Mirroring Disks
- Disk Scheduling
- Prefetching and Buffering

I/O Model of Computation

- In trying to estimate performance of DBMS:
  - Assume database is too large to fit in memory
  - Key parts buffered
  - Many users (so disk controller needs to queue requests)
  - Disk requests will be for data on a random location on drive.
- Most importantly, we assume the **I/O Model of Computation**: That is, the time required to move a block to or from main memory is much greater than any time spent manipulating the data in memory. So the latter time will be ignored in our estimates

Sorting in Secondary Storage

Example: Assume have a table R consisting of 50,000,000 rows. Assume block size is 16KB (2^{14} bytes) and 100 records fit in a block. i.e., 1 record is roughly 160 bytes.

- So R fits in 500,000 blocks.
- On the other hand, if have 500MB of memory. Then can at most hold $500*2^{20}/2^{14} = 32{,}000$ blocks.
- So if want to sort on some field can't read everything into RAM

Bottom up Merge Sort

- Common main memory sort algorithm
- Work in passes. In pass i merge adjacent run of length 2^i into runs of length 2^{i+1}

  Ex 4 2 5 1 6 3 (runs length 1)

  2 4 1 5 3 6 (runs length 2)

  1 2 4 5 3 6 (runs length 4)

  1 2 3 4 5 6 (runs of length 8) -- done
- So have about $\log_2 n$ passes and each pass is O(n) work. So O(n*$\log_2$ n) time algorithm.

Two Phase Multiway Merge-Sort (TPMMS)

- Phase I -- read in main memory sized (M bytes) chunks from the table to be sorted. This gives M/B blocks. Sort these according to the desired field. Save these sorted sub-lists back to disk.
- Phase II -- merge these sub-lists together to produce one sorted list. We will explain how this is done in a moment.

Phase I Example

- Using the R from a couple slides ago…
  - R was 500,000 blocks long and main memory holds 32,000 blocks.
  - So would make 500000/32000 = 16 sub-lists.
  - Each block from R needs to be read once from disk and written once to disk for a total of 1,000,000 I/Os.
  - If an I/O takes 1/100 of second. This would take 10,000 seconds of 10000/3600 = 2.7 hours

Phase II

- To do this phase we could do something like in the main memory case--- work in passes, read a block each from pairs of ``adjacent'' sorted sub-lists, merge these, and write to a twice as long output sub-list. Do this for all sub-lists. After $\log_2 n$ passes we would be done.
- This is somewhat inefficient, since only have two blocks from the file in memory at a time.
- Instead, we read from M/B-1 different sub-lists one block. We merge these into one output block. When it fills write it disk and start filling it again.

Phase II -cont'd

How much more efficient is this last idea than just merging two blocks?

- Well, now it takes $\log_{M/B - 1} n$ passes rather than $\log_2 n$ passes.
- As each pass can take a while this can mean significant savings.

Accelerating Access to Secondary Storage

- In the case of small transactions it is hard to do anything about the organization of blocks on the disks in order to speed up the time per I/O.
- On the other hard, if we are doing one big thing like sorting we can do things to reduce the time per I/O. For example:
  - place blocks that are accessed together on the same cylinder
  - divide the data over several drives so can have more drive head assemblies working independently of each other
  - Mirror Disks
  - Use a more clever disk scheduling algorithm
  - Prefetch blocks to main memory in anticipation of their use.

Organizing Data by Cylinder

- Seek time takes on average about half the time used to find a block on the disk.
- By storing data that is likely to be accessed together in adjacent blocks we cut down on this time.
- Example: suppose a cylinder stores 500 blocks. Time to read adjacent blocks 1/200 of a second (rather than 1/100 each). So can read a whole cylinder in 2.5 s. R takes 500,000/500 cylinders = 1000. So can read R in 2.5*1000 + 1/100*1000 = 2510sec. So Phase I would take 5020 sec
- 1/100*1000 is the time it takes to move between cylinders

Disk Scheduling

- A simple but effective way to schedule a large number of disk requests is to use the **elevator algorithm**.
- Basically, the head roughly moves from the inner cylinder to the outer cylinder and then back down to the inner cylinder, etc.
- As it passes a cylinder the head notices whether there has been a request for blocks on that cylinde. If so, it reads or writes those blocks.
- Once done a cylinder, the head is moved to the next cylinder in the direction it was traveling
- If there were no further requests in that direction, the direction of moving inward or outward is reversed.
- The book gives an example showing this is about 2 to 3 times more efficient then doing things first come first serve

Prefetching (aka double buffering)

- In some applications one can predict the order in which blocks will be requested from disk.
- If so, one can load them into memory before they are needed.
- This can make it easier to schedule the disk.
- In sorting, if only have 16 sorted sub-lists we could could have a buffer of two blocks for each sub-list. We could be reading into one block while using the other for merging.