Two Pass Algorithms

CS157B
Chris Pollett
Mar. 16, 2005.

Outline

- More Sort Based Algorithms
- Two Pass Algorithms Based on Hashing
- Indexed Based Algorithms
- Parsing

Grouping and Aggregation using Sorting

- To do grouping and aggregation:
  - Read the records into memory, M blocks at a time. Sort M blocks, using the grouping attributes as the sort key, Write each sorted list to disk.
  - Use one main buffer for each sublist, and initially load the first block of each sublist into its buffer.
  - Repeatedly find the least value, v, of the (grouping attributes among the first available tuple in the buffer. For this value, we (a) compute all aggregates for the group v belong to. (b) if a buffer becomes empty replace it with the next buffer from the same sublist.
  - If no more tuples for this v go on to next smallest v.
  - Notice take 3B(R) time and works if $B(R) < M^2$

Sorting and Unions, Intersections, etc.

- Do Phase I of sort for both R and S. (Sort into subfiles of M blocks). Assume total number of subfiles generated <M-1.
- In Phase II of sort compute the desired Union, intersection at the same time we are doing the merge phase.
- This takes 3(B(R)+B(S)) I/Os and the phas II step works provided $B(R)+B(S) \le M^2$.

Sort-based Join

- Want to Join R(X,Y) and S(Y,Z)
- Create sorted sublists of size M, using join attribute Y for both R and S.
- Bring the first block of each sublist into a buffer (assume <= M sublists).
- Repeatedly find the least Y value y amoung the sublist blocks in memory. Identify tuples in both relations having this Y value. Output the join of all these tuples.
- Take time 3(B(R) +B(S)) provided B(R)+B(S) $<M^2$

Partitioning Relations by Hashing

- For the algorithms we'll consider next, it is useful to be able to read a file quickly into a hash table.
- To do this we use one block of main memory to read from the file and use M-1 blocks for each bucket of the hash table (we assume this has at most M-1 buckets).
- We read successive blocks from the file apply the hash function to each tuple in these blocks and move these tuples to the correct bucket buffer.
- When a bucket buffer gets filled we write it to disk.

Hashed based Duplicate Elimination

- First hash file into bucket files.
- Then eliminate duplicates from each bucket file using the one pass algorithm and output the results.
- The idea is if t occurs multiple times, it will always hash to the same bucket. So can eliminate duplicates bucketwise.
- This works provided hash function is random-like and B(R) is less than M^2. It takes 3B(R) time.

Hash-Based Algorithm for Grouping and Aggregation

- Want to do grouping and aggregation on R.
- As with hash based duplicate elimination, the first step is to hash R into a hash file.
- Want a hash function which only depends on the grouping attributes.
- Now use the one pass algorithm for aggregation for each bucket.
- So need each bucket is $<M$.
- So works if $B(R) < M^2$. It takes $3B(R)$ time to do this operation.

Hash based and Unions, Intersections, etc

- Hash both R and S to hash tables.
- Apply one pass algorithm to each of the bucket 1's of R and S. Then the one pass algorithm to the bucket 2's,… etc.
- Works provided $\min(B(R), B(S)) < M^2$.
- Takes time $3(B(R)+B(S))$.

Hash-Join

- Use only the join attribute as the hash key.
- Hash R and S to hash tables.
- Do one pass join of bucket 1's, 2's, etc.
- Works provided $\min(B(R), B(S)) < M^2$.
- Takes time $3(B(R)+B(S))$.

Index-based Selection

- Suppose have a clustering index and want to do $\sigma_{a=v}(R)$.
- Then the number of I/Os will be roughly B(R)/V(R,a).
- Here we are ignoring cost of index, round off errors, and that the blocks of R might not be completely filled with tuples from R.
- If R is a non-clustering index, then the cost will be approximately T(R)/V(R,a). So it will be more likely a table scan will be faster.

Index-based Join

- Examine each block of R. For each tuple t in that block, look up all things that join with it in S and output the joins.
- Cost will approximately be $T(R)T(S)/V(S,Y)$, where Y is the join attribute if S is nonclustered and $T(R)B(S)/V(S,Y)$ if it is clustered.

Parsing Queries

- We now begin to discuss how queries are parsed. Below is a rough flow of the operations: