Lecture 11: Sort and Join Algorithms

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Sort

1. We need sorting because in the relation model, tuples in a table have no specific order
2. Sorting is used in ORDER BY, GROUP BY, and DISTINCT
3. If data fits in memory, then we can use a standard algorithm like **quicksort**
4. If data does not fit, we need to use external sorting

External Merge Sort

1. Sorting phase: Sort small chunks of data that fit in main memory, and then write back to disk
2. Merge phase: Combine sorted subfiles into a larger single file
3. Algorithm for two way merge sort
   (a) Pass 0: Reads every B pages of the table into memory. Sorts them, and writes them back into disk. Each sorted set of pages is called a **run**
   (b) Pass 1,2,3...: Recursively merges pairs of runs into runs twice as long.
4. Number of passes: $1 + ceiling(log_2(N))xw$
5. Total I/O cost: $2N*$ (# of passes)
6. General algorithm (K-way merge)
   (a) Pass 0: Use B buffer pages, produce N/B sorted runs of size B
   (b) Pass 1,2,3...: Merge $B - 1$ runs
   (c) Number of passes = $1 + ceiling(log_{B-1}(N/B))$
   (d) Total I/O cost: $2N*$ (# of passes)

Sorting with B+ Tree

1. We can accelerate sorting using a **clustered** B+ tree by simply **scanning the leaf nodes from left to right**
2. Bad idea using an **unclustered** B+ tree to sort because it causes a lot of I/O reads (random access through pointer chasing)

Alternatives to Sorting

1. We can remove duplicates using a hash table
2. Hashing can be computationally cheaper than sorting

Importance of Join

1. Unnecessary repetition of information must be avoided
2. We decompose tables using normalization theory
3. Joins are used to reconstruct original tables
4. Joins are very common, and must be heavily optimized
5. Cartesian products are rarer, but are very computationally expensive

Simple Nested Loop Join

1. Nested for loop iterating over tuples in both tables, if the tuples match the join predicate, then output them
2. Bad because for every single tuple in one table, you scan every tuple of the other table
3. Optimization: Using the smaller table as the outer table

Block Nested Loop Join

1. Algorithm: reads and compares blocks at a time
2. Algorithm performs fewer disk access because we scan the second table for every block instead of for every tuple
3. Optimization: Using the smaller table as the outer table reduces the number of I/O
4. You can significantly reduce the cost by using multiple buffers

Index Nested Loop Join

1. Basic nested loop joins are bad because we have to do a sequential scan to check for a match in the inner table
2. **We can use an index to find inner table matches**

General Nested Loop Join: Summary

1. Pick the smaller table as the outer table
2. Buffer as much of the outer table in memory as possible
3. If possible, leverage an index to find matches in inner table

Sort-Merge Join

1. Sort phase: First sort both input tables on the join attribute
2. Merge phase: Scan the two sorted tables in parallel, and emit matching tuples
3. This algorithm is very useful if one or both tables are already sorted on join key
4. Worst case during join phase: The join attribute of all the tuples in both relations contain the same value (**very unlikely**)

Costs of Join Algorithms

1. For tables R and S:
   (a) M pages in R, p_r tuples per page, m tuples total
   (b) N pages in S, p_s tuples per page, n tuples total

JOIN ALGORITHM	I/O COST	TOTAL TIME
Simple Nested Loop Join	M + (m·N)	1.3 hours
Block Nested Loop Join	M + (M·N)	50 seconds
Index Nested Loop Join	M + (m·log N)	20 seconds
Sort Merge Join	M + N + (sort cost)	0.75 seconds