CSE 444: Database Internals

Lecture 8
Operator Algorithms (part 2)

Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)

Review: Access Methods

- **Heap file**
  - Scan tuples one at the time
- **Hash-based index**
  - Efficient selection on equality predicates
  - Can also scan data entries in index
- **Tree-based index**
  - Efficient selection on equality or range predicates
  - Can also scan data entries in index

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- V(R, a) = # of distinct values of attribute a
- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)
- Note: we ignored I/O cost for index pages

Index Based Selection

- Example: B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R) = ?$

- Table scan: B(R) = 2,000 I/Os
- Index based selection
  - If index is clustered: B(R)/V(R,a) = 100 I/Os
  - If index is unclustered: T(R)/V(R,a) = 5,000 I/Os
- Lesson
  - Don't build unclustered indexes when V(R,a) is small !

Index Nested Loop Join

R ⋈ S

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Cost:
  - If index on S is clustered: B(R) + T(R)B(S)/V(S,a)
  - If index on S is unclustered: B(R) + T(R)T(S)/V(S,a)

Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)

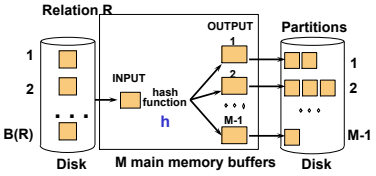
Two-Pass Algorithms

- What if data does not fit in memory?
- Need to process it in multiple passes

- Two key techniques
  - Hashing
  - Sorting

Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. B(R) <= M^2

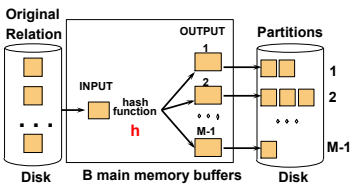
Partitioned (Grace) Hash Join

R ⋈ S

- Step 1:
  - Hash S into M-1 buckets
  - Send all buckets to disk
- Step 2
  - Hash R into M-1 buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

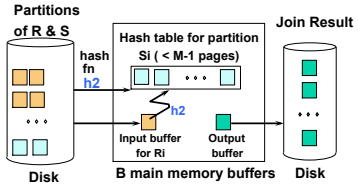
Partitioned Hash Join

- Partition both relations using hash fn **h**
- R tuples in partition i will only match S tuples in partition i.



Partitioned Hash Join

- Read in partition of R, hash it using h2 (≠ h)
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase



Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

Partitioned Hash Join

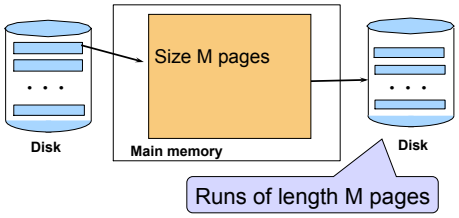
- See detailed example on the board

External Sorting

- Problem: Sort a file of size B with memory M
- Where we need this:
  - ORDER BY in SQL queries
  - Several physical operators
  - Bulk loading of B+-tree indexes.
- Sorting is two-pass when B < M^2

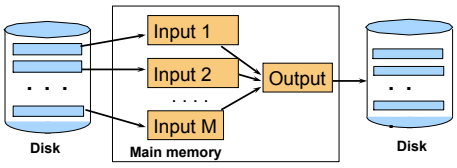
External Merge-Sort: Step 1

- Phase one: load M pages in memory, sort



External Merge-Sort: Step 2

- Merge M – 1 runs into a new run
- Result: runs of length M (M – 1)≈ M^2



If B <= M^2 then we are done

External Merge-Sort

- Cost:
  - Read+write+read = 3B(R)
  - Assumption: B(R) <= M^2
- Other considerations
  - In general, a lot of optimizations are possible

External Merge-Sort

- See detailed example on the board

Two-Pass Join Algorithm Based on Sorting

Join R ⋈ S

- Step 1: sort both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Step 2: Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Total cost: 5B(R)+5B(S)
- Assumption: B(R) <= M^2, B(S) <= M^2

Two-Pass Join Algorithm Based on Sorting

Join R ⋈ S

- If B(R) + B(S) <= M^2
  - Or if use a priority queue to create runs of length 2|M|
- If the number of tuples in R matching those in S is small (or vice versa)
- We can compute the join during the merge phase
- Total cost: 3B(R)+3B(S)

Two-Pass Join Algorithm Based on Sorting

- See detailed example on the board

Summary of Join Algorithms

- Nested Loop Join: B(R) + B(R)B(S)
  - Assuming page-at-a-time refinement
- Hash Join: 3B(R) + 3B(S)
  - Assuming: min(B(R), B(S)) <= M^2
- Sort-Merge Join: 3B(R)+3B(S)
  - Assuming B(R)+B(S) <= M^2
- Index Nested Loop Join: B(R) + T(R)B(S)/V(S,a)
  - Assuming S has clustered index on a

Summary of Query Execution

- For each logical query plan
  - There exist many physical query plans
  - Each plan has a different cost
  - Cost depends on the data
- Additionally, for each query
  - There exist several logical plans
- Next lecture: query optimization
  - How to compute the cost of a complete plan?
  - How to pick a good query plan for a query?