15-823
Advanced Topics in Database Systems Performance

Query Evaluation Techniques for Large Databases

Graefe on Query Evaluation

- A survey of query evaluation techniques:
  - sorting
  - hashing
  - disk access
  - aggregation
  - duplicate removal

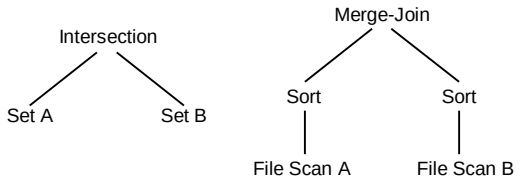
Overview/Main Points

- Architecture of query engines
- Sorting
- Hashing
- Disk Access
- Aggregation/duplicate removal

Architecture of query engines

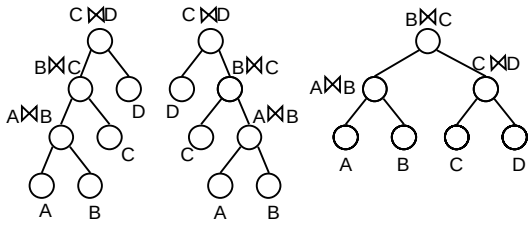
- Query processing algorithms iterate over input sets
  - Logical algebra, i.e., relational algebra
  - Physical algebra: implementation, algorithms…
- Optimization is based on physical algebra
- Operators (a.k.a., *iterators*)
  - Temp files, buffers
  - Same computer and IPC
  - Operator = *open, next, close* + function calls
- Key: Synchronization / transfer between operators
- Query plans

Physical/Logical Algebra Example



- 1 LA operator = many PA operators
- Some PA's have no LA counterpart (e.g., sort)
- PA bias towards inputs of commutative operations (NL)

Query Plans



Left-deep Right-deep Bushy

…or DAG for queries with common subexpressions

Sorting

- Sorting algorithms merge sorted runs
- Interface same as for iterators
- Exploit duality of quicksort and mergesort.
  - Mergesort: Divide physically, combine logically
  - Quicksort: Divide logically, combine physically
  - Fan-in
- Runs: Quicksort and replacement-selection
  - Level-0 runs are merged into level-1 runs

Replacement Selection

- Priority heap: “insert”, “remove smallest”
- Worst case scenario: reverse order
- On the average (random): 1 run=2*memory_size
- Half #of runs, runs twice as long quicksort (good)
- Kills it in memory management
  - No memcpy – keep pages around
  - Halves the heap capacity
  - Wipes out run length advantage
- Solution: copy records into holding space
  - Messy with variable-length attributes

Merging Runs

- Level 0 runs are merged into Level 1 runs, etc…
- Buffer space for each input run and merge output
- I/O in *clusters.*
- Efficiency issues:
  - read-ahead/write-behind w/ double buffering
  - large cluster sizes for run files
  - # of runs is usually not a power of fan-in
  - Divide memory among multiple final merges
  - specify separately final fan-in and normal fan-in

Hashing

- For equality matches, only…
- Hash table overflow
  - Avoidance – partition conservatively, tune later
  - Resolution – start hashing, partition dynamically
  - Hybrid does both using up all the memory
- Assigning hash buckets to partitions
- A good hash function is key for performance!

Hash Buckets To Partitions

- Overflow => fixed # buckets in new partition
  - A priori decided # of partitions (fan-out)
  - Gamma
- Bucket tuning and dynamic destaging (next slide)
- Gather statistics before hashing and use them
- Recursive partitioning

Tuning / Dynamic Destaging

- Many small partitions, collapse later
- Upon overflow write largest partition to disk
- Hopefully, small “mergeable” ones remain in memory

+ Effective for skew in the first input
+ Probe immediately from second input

(which input should we assign the smaller relation to?)

– No effect for skew in second input
– Cluster size must be small => lots of I/O operations

Disk Access

- File scans – read ahead
  - track-at-a-crack (prefetch N full + 1 empty page)
  - Extents – may need to bypass operating system
- Indices
  - B-trees, B+-trees, B*-trees
  - Argue that leafs should *point* to records
    - Scan index for values in the index
    - Join indices to answer query, or take union/intersection
    - Join tables = join indices and find records
- Buffer management

Aggregation/duplicate removal

- Scalar aggregate (e.g., max)
  - calculates single scalar value from unary input
  - Can be computed in single pass
- Aggregate functions (e.g., group-by)
  - Determine set of values from a binary input
  - e.g., sum of salaries from each department
  - Require *grouping*
- Duplicate removal treated like aggregate functions

Aggregation Algorithms

- Nested loop
  - For each input, aggregate into or append to output file
- Sorting
  - Detect and remove duplicates during run creation
  - “early” aggregation, speeds up final merging
  - Optimizations same as before, e.g., read-ahead etc.
- Hashing
  - Can only do aggregation for in-memory partitions

Performance Notes

- Early aggregation pays off because
  - It reduces run size
  - It makes merging faster
  - sort, hash => logarithmic performance on input size
- Multilevel aggregation?
  - sum(salary by emp.id by emp.dept by emp.division)
  - Hard to avoid multiple joins
- Real-time systems
  - Trade between precision and response time

…Some Course Corrections?

- Optimization + Processing + Physical Design
  - (P.D. = set of useful indexes)
- Trade-off between compilation and interpretation
  - Compilation is faster, but one doesn't have all the info

Optimization Issues

- Unknown "beforehand" information
  - Selectivity estimation
  - Statistics/histograms
  - Run-time parameter values
- Therefore, run-time optimization
  - Should not be more costly than recompilation
- Cost function accuracy
  - Changes on resource availability?
  - Papers often do not deal with multi-operator reality
  - Real systems do not incorporate proposed solutions

Neglected Issues

- Nested iterations
  - Same nested operations reuse resources
  - Different ones compete for resources
  - A conceptual model (simpler than SQL) is needed
- Indexed views
  - Materialized views are a great performance booster
  - Building up on redundancy has unsolved problems
  - Index views along with tables
- Caching recent queries

Performance

- Metrics – use them from the correct perspective
- Predictability is more important
- Index tuning and heuristics
- Research for orders of magnitude