Query Optimization: Exercise

Session 14

Bernhard Radke

February 5, 2018

Motivation

- declarative query has to be translated into an imperative, executable plan
- usually multiple semantically equivalent plans (search space)
- possibly huge differences in execution costs of different alternatives

Goal: find the cheapest of those plans

Query Graph

- undirected graph
- nodes: relations
- edges: predicates/joins
- different shapes (e.g. chain, star, tree, clique)
- shape influences size of the search space

Join Tree

- inner nodes: operators (e.g. join, cross product)
- leaves: relations
- different shapes
  - linear (left-deep, right-deep, zigzag)
  - bushy
- desired shape influences size of the search space
  - with cross products: number of tree shapes * number of leaf permutations
  - without cross products: depends on the shape of the query graph

Selectivity, Cardinality

$$f_p = \frac{|\sigma_p(R)|}{|R|}$$

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

Costs

$$C_{out}(R) = 0$$
$$C_{out}(R_i \bowtie R_j) = |R_i \bowtie R_j| + C_{out}(R_i) + C_{out}(R_j)$$

- more advanced cost functions for different physical join implementations
- properties
  - symmetry: $C(A \bowtie B) = C(B \bowtie A)$
  - ASI: rank function r such that $r(AUVB) \leq r(AVUB) \Leftrightarrow C(AUVB) \leq C(AVUB)$

Greedy Heuristics

- choose each relation as start node once
  - greedily pick adjacent nodes to join such that a specific function (e.g. MinSel) is minimized/maximized
- pick the cheapest tree
- produces linear trees

Greedy Operator Ordering (GOO)

- greedily pick edges such that the intermediate result is minimized
- merge nodes connected by the picked edge
- calculate cardinality of merged node
- calculate selectivities of collapsed edges (product of individual selectivities)
- can produce bushy trees

Maximum Value Precedence (MVP)

- heuristic: prefer to perform joins that reduce the input size of expensive operations the most
- algorithm builds an effective spanning tree in the weighted directed join graph (edges and nodes have weights)
  - physical edge: $w_{u,v} = \frac{|\bowtie_u|}{|u \sqcap v|}$
  - virtual edge: $w_{u,v} = 1$
  - node: $w(p_{i,j}, S) = \frac{|\bowtie^S_{p_{i,j}}|}{|R_i \bowtie_{p_{i,j}} R_j|}$
- take edges with weight < 1 (reduce work for later operators)
- add remaining edges (increase input sizes as late as possible)

IKKBZ

- generates optimal left deep trees for acyclic queries in polynomial time (requires cost function with ASI property)
- for each relation R in the query graph
  - build the precedence graph rooted in R
  - find subtree whose children are chains
  - build compound relations to eliminate contradictory sequences (normalize)
  - merge chains (ascending in rank)
  - repeat until the whole join tree is a chain
  - denormalize previously normalized compound relations
- pick the cheapest of all generated sequences

Dynamic Programming

- optimality principle
- construct larger trees from optimal smaller ones
- try all combinations that might be optimal
- different possibilities to enumerate sets of relations
  - DP_{size}: enumerate sets ascending in size
  - DP_{sub}: enumerate in integer order
  - DP_{ccp}: enumerate connected component complement pairs
    - adapts to the shape of the query graph
    - lower bound for all DP algorithms
  - DP_{hyp}: handles hypergraphs (join predicates between more than two relations, reordering constraints for non inner joins, graph simplification)

Memoization

- recursive top-down approach
- memoize already generated trees to avoid duplicate work
- might be faster, as more knowledge allows for more pruning
- usually slower than DP

Transformative Approaches

- apply equivalences to initial join tree
- makes it easy to add new equivalences/rules (in theory)
- use memoization (keep all trees generated so far)
- naive implementation generates a massive amount of duplicates
- duplicates can be avoided by disabling certain rules after a transformation has been applied (introduction of new rules becomes harder)

Permutations

- construct permutations of relations (left deep trees)
- choose each relation as start relation once
  - successively add a relation to the existing chain (recursively enlarge the prefix)
  - only explore the resulting chain further if exchanging the last two relations does not result in a cheaper chain
  - recursion base: all relations are contained in the chain $\Rightarrow$ keep chain if cheaper than cheapest chain seen so far
- any time algorithm (can be stopped as soon as the first complete permutation is generated)
- finds the optimal plan eventually

Random Join Trees (uniformly distributed)

general approach:

- set of alternatives S
- count number of alternatives $n = |S|$
- bijection $rank : S \rightarrow [0, n[$
- draw a random number $r \in [0, n[$
- $rank^{-1}(r)$ gives a random element from S (unranking)

implementation

- random permutation (left deep tree, leaf labeling)
- random tree shape (Dyck words)
- random trees without cross products for tree queries (pretty complex)

Quick Pick

- generate pseudo random trees
- randomly pick an edge from the query graph
- no longer uniformly distributed $\Rightarrow$ no guarantees
- use union-find datastructure to identify subsets containing the nodes connected by an edge

Meta Heuristics

- universal optimization strategies
- Iterative Improvement
  - start with random join tree
  - apply random transformation until minimum is reached
  - might be stuck in local minimum
- Simulated Annealing (inspired by metallurgy)
  - start with random join tree
  - apply random transformation
  - accept transformed tree either if it is cheaper or - with a temperature dependent probability - even if it is more expensive
  - decrease temperature over time
  - allows to escape local minima

Meta Heuristics

- Tabu Search
  - start with random join tree
  - investigate cheapest neighbor even if it is more expensive
  - keep (recently) investigated solutions in tabu set to avoid running into circles
- Genetic algorithms
  - population of random join trees
  - simulate crossover and mutation
  - survival of the fittest

Combinations and Hybrid Approaches

- Two Phase Optimization: II followed by SA
- AB Algorithms: IKKBZ followed by II
- Toured Simulated Annealing: run n times with different initial join trees (e.g. results of GreedyJoinOrdering-3)
- GOO-II: Run II on the result of GOO
- Iterative DP (IDP-1): build join trees with up to k relations, replace cheapest with compound, repeat

Order Preserving Joins

- non-commutative operators
- how to parenthesize the chain?
- maintain arrays p (predicates), s (statistics), c (costs), t (split positions)

Accessing the Data

- Yao, Cheung: estimate the number of pages to be read from disk if we want to read k (distinct) tuples directly
- Bitvector: estimate sequential disk access costs for a sequence of k tuples sorted in the order they reside on disk
- Selectivity estimation (Histograms)

- Slides and exercises: db.in.tum.de/teaching/ws1718/queryopt
- Send any questions, comments, solutions to exercises etc. to radke@in.tum.de
- Exam on 27th of February at 13.30 in 102 Interims Hörsaal 2.

Good Luck!