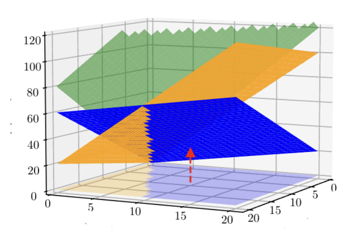
CS286 2024 Hellerstein

Lecture 10: LIP, Semi-Joins + Adaptivity, Robustness

Context

1. More AdaptiveQP
    a. Survey article on Adaptive QP from 2007
    b. Less aggressive techniques than eddies, easier to integrate into Selinger/Cascades.
        i. MidQuery Reoptimization (Kabra/DeWitt):
            - If we materialize a subexpression (a "stage"), can re-optimize the rest
            - Can "force" a materialization point if we like during optimization
        ii. IBM "progressive query optimization" (POP): use streaming cardinality check to abort a "stage"
            - CHECK operator w.r.t. optimizer's original guess
        iii. LIP from Wisconsin (2017): *today's reading*
    c. SkinnerDB from Cornell (2019)
        i. RL approach to quickly try many different left-deep trees
        ii. (RL + Eddies had been done previously)
2. Parametric Query Optimization
    a. We have a cost formula for *every* plan as a function of its parameters (e.g. size of relation 1, size of relation 2, selectivity of predicate 1, etc). If this cost formula is linear, this is a hyperplane
    b. Imagine we *store all these hyperplanes,* one per plan choice, in some kind of "index" (a convex polytope)
    c. Then we "query" this index for a given set of parameters by finding the lowest-cost plane at that setting of the parameters
    d. (Tiemo Bang uses this general idea in his Cloud Oracles work)



3. Robust Q.O.

    a. IBM POP was designed for robustness: *validity range* of a selectivity estimate: outside this, the plan is suboptimal
        i. Based on *parametric query optimization*
    b. Robust cost estimation, using sampling and confidence bounds to control errors of estimators.
    c. Plan Bouquets instead of cost estimation to minimize Maximum SubOptimality
    d. More references in LIP paper
4. Techniques
    a. SemiJoins commonly used for distributed databases
    b. Yannakakis' Algorithm is a technique to evaluate a large class of "easy" queries (technically, *acyclic conjunctive queries*) in time polynomial in the size of the query, input and output. (Applies to set-oriented relations!)
        i. apply semijoins to each input relation so that it has no "dangling" tuples that have no impact on the output. (this is called a "full reducer")
            - May be >1 semijoin per input relation if it joins on >1 attribute!
        ii. construct a join tree, push down projections as far as possible
        iii. then each intermediate result will be O(|input|*|output|).

    Proof: Consider some intermediate subtree $T = (S_1 \bowtie \ldots \bowtie S_n) \bowtie R$. Replace with $T' = \pi_{Y \cup Z}(T)$ where Y is the set of attributes in R, and Z is the set of attributes in the output other than Y. Clearly $T' \subseteq \pi_Y(T) \times \pi_Z(T)$. Now, $|\pi_Y(T)| \leq |R|$ (input) and $\pi_Z(T) \leq |output|$.
    c. Simple independent Job Shop scheduling
        i. Imagine you have n independent *expensive filters*, each with $cost_i$ *and* $selectivity_i$.
        ii. Adjacent Sequence Interchange property: swap adjacent pairs to be ordered by *rank* improves cost
            - $rank_i = (selectivity_i - 1)/cost$
            - Generally, the optimal order: increasing *rank*
    d. Ibaraki-Kameda: apply this to left-deep join trees. Consider R join S join T
        i. for each leftmost table
            - remaining "half-joins" are like selections (selectivity may be >1)
            - order the half-joins by rank
            - save the result
        ii. Chose the leftmost rank-ordered plan that's cheapest
    e. Krishnamurthy-Boral-Zaniolo:
        i. Lots of redundant work in IK: two trees will share common "upstream suffixes"
        ii. Removes this to get from O($n^2 logn$) to O(n^2) algorithm

LIP

Stated goals: *robust* and *good* join order selection.

Focus: star schema

- One big Fact table, many Dimension tables
    - Key-Foreign Key relationships
- Queries are mostly left-deep trees
- The perfect setting for IK!
    - No need to choose a spanning tree or outermost table!

QP Tricks

1. For each dimension table D dynamically precompute $\text{BloomFilter}_{D.key}$
    a. Cost is negligible relative to building hashtables
2. Rewrite query to include additional fact-table UDF selections based on Bloom filters
    a. WHERE BloomCheck($F.fkey_D$, $\text{BloomFilter}_{D.key}$)
3. Do an eddy-like thing on the selections
    a. Stream data, dynamically learn the selectivities and reorder while applying them
    b. Work on tuple *batches* to keep bloom filter in cache for a while
        - batch_size = 64
        - while not done:
            - For each batch, run through current bloom filter order
                a. Track *result_batch, count[f], miss[f]* for each filter *f*
            - re-sort filters by selectivity
            - merge *result_batch* into *results*
            - double the batch size
    c. Selections are easy to reorder relative to join: stateless, rank-ordered as in Predicate Migration
    d. Will converge quickly (assertion: 3-4 doublings of batch size)
        - sampling bounds like Chebyshev
        - could work harder to bound confidence of each filter (explore/exploit tradeoff)

Why doesn't this require scanning tables multiple times?

- Going to build hashtables on inners anyway
- Adaptive QP on outer a la eddies

Bloom Filter Fun

- Note: BloomFilter merge is OR, which is associative/commutative/idempotent
    - Forms a lattice
    - Trivially parallelizable
- Empirical analysis:
    - identity hash function (w/modulus) worked fine
    - ~8 bits (1 Byte) of Bloom filter per tuple of input
        - Goal: fit a Bloom filter in processor cache
        - L1: 128 KB on an Apple M2. I.e. 128k tuples in bloom filter
        - L2: 16MB on an Apple M2. I.e. 16M tuples

Cost/Robustness Analysis

Start without LIP:

- Build costs are independent of order, so focus on Probes
- First n terms of [*geometric series*]() $\Sigma ar^n = a*(1-r^n)/(1-r)$

$$\begin{aligned} s_n &= ar^0 + ar^1 + \cdots + ar^{n-1}, \\ rs_n &= ar^1 + ar^2 + \cdots + ar^n, \\ s_n - rs_n &= ar^0 - ar^n, \\ s_n(1-r) &= a(1-r^n), \\ s_n &= a\left(\frac{1-r^n}{1-r}\right), \text{ for } r \neq 1. \end{aligned}$$

  -

- Now can bound cost of any plan
  - $a = |F|$
  - $n = \sigma_{min}$ for lower bound, $n = \sigma_{max}$ for upper bound
- Could be a big range: Formula (7)

$$T(P_w) - T(P_b) = \sum_{i=1}^{n-1} \left(\sigma_{n'} \ldots \sigma_{(n-i+1)'} - \sigma_{1'} \ldots \sigma_{i'}\right)|F| \quad (7)$$

  - Simplify in terms of $\sigma_{max} - \sigma_{min} = \sigma_{n'} - \sigma_{1'}$

$$\begin{aligned} \sigma_{n'}\sigma_{(n-1)'} \ldots \sigma_{(n-i+1)'} - \sigma_{1'}\sigma_{2'} \ldots \sigma_{i'} &\geq (\sigma_{n'} - \sigma_{1'})\sigma_{2'} \ldots \sigma_{i'} \\ &\geq (\sigma_{n'} - \sigma_{1'})\sigma_{1'}^{i-1} \end{aligned}$$

  - Plugging into (7) we get Formula (8):

$$\begin{aligned} T(P_w) - T(P_b) &\geq \sum_{i=1}^{n-1} \sigma_{1'}^{i-1}(\sigma_{n'} - \sigma_{1'})|F| \\ &= \frac{1 - \sigma_{\min}^{n-1}}{1 - \sigma_{\min}}(\sigma_{\max} - \sigma_{\min})|F| \qquad (8) \end{aligned}$$

A definition of Robustness:

- θ-fragile: diff between worst and best is at least θ
- $\boldsymbol{\Theta}$-robust: diff between worst and best is at most $\boldsymbol{\Theta}$

normalized by |F| and the spread of selectivities:

$$\theta \leq \frac{T(\mathcal{E}_w) - T(\mathcal{E}_b)}{(\sigma_{max} - \sigma_{min})|F|} \leq \Theta, \qquad \sigma_{max} \neq \sigma_{min} \qquad (9)$$

Wrapping our head around this:

- high fragility (θ) means the worst plan is AWFUL.
- low robustness ($\boldsymbol{\Theta}$) means worst plan is close to optimal
- Now compare two optimization schemes O_1 and O_2
  - If $\boldsymbol{\Theta}$ for O_1 is less than θ for O_2, O_1 is the clear winner
  - O_1 is LIP, O_2 is non-LIP

Why this normalization?

- |F|? *a "per-tuple" definition*
- (σ_{max} - σ_{min})? *Compare robustness of optimizer schemes in a query-independent way*
- Assumptions here?

Robustness of LIP:

$$\mathrm{T}(B_w) - \mathrm{T}(B_b) \leq \frac{1}{2}\sigma_1\sigma_2...\sigma_n\epsilon n(n+1)\left[\frac{1}{\sigma_{\min}} - \frac{1}{\sigma_{\max}}\right]|F| \quad (16)$$

Key Result: From Equation 16, it is clear that LIP with adaptive reordering is a Θ-robust evaluation strategy, for

$$\Theta = \frac{1}{2}\frac{\sigma_1\sigma_2...\sigma_n}{\sigma_{\min}\sigma_{\max}}\epsilon n(n+1) \quad (17)$$

-
- (17) follows:
  $\boldsymbol{\Theta}_{LIP} = 1/2\sigma_1\sigma_2\ldots\sigma_n\epsilon n(n+1)|F| \cdot (\sigma_{max}-\sigma_{min})/\sigma_{min}\sigma_{max} \cdot 1/|F|(\sigma_{max}-\sigma_{min})$
  $= ½ \cdot \sigma_1\sigma_2\ldots\sigma_n/\sigma_{min}\sigma_{max} \cdot \epsilon n(n+1)$
- Recall that without LIP we had:
  $\theta_{NoLIP} = (1 - \sigma_{min}^{n-1})/(1 - \sigma_{min})$
- Messy assertion:
  From this discussion, it is clear that LIP *theoretically guarantees robustness, whereas the naive evaluation strategy is likely to make plan selection much more fragile.*
  - What should this say? When is $\boldsymbol{\Theta}_{LIP} <= \theta_{NoLIP}$?

Evaluation Study

- Tune the Bloom filters
- Study how LIP does relative to all possible orders for independent predicates (nice)
  - Small dimension rows though: (integer, char, char)
- Model correlations. I was confused by the description here, and not convinced.
  - I'd like to see an adversarial correlation workload. How wrong can LIP be?
    - Need to drive correlation across *predicates*
    - E.g. conditional prob:
      - if column x = TRUE, the best plan is filter 1, 2, 3, 4, 5, 6, … n
      - if column x = FALSE, the best plan is filter n, n-1, …, 3, 2, 1
  - Then I'd like to understand how we can turn knobs on data to induce a spectrum from worst to best scenario for LIP

Implemented in Quickstep (acquired by Pivotal) and SQLite!

- See SQLite: Past, Present and Future
- *"SQLite's query planner uses a straightforward model to determine whether a Bloom filter should be constructed. For each inner table, the query planner generates the [LIP] Bloom filter logic if all of the following conditions are true:*
  a. The number of rows in the table is known by the query planner.
  b. The expected number of searches exceeds the number of rows in the table.

c. *Some searches are expected to find zero rows.”*

A Host of Questions

- On the limitations of LIP
  - Correlation study: do you believe it? Adversarial data?
  - Beyond star schemas: Adversarial queries?
- How might we integrate LIP with more workloads
  - Can we predict early that LIP may fail?
  - Can we do LIP on subqueries?
  - Access method alternatives?
  - Integration with Cascades?
  - Integration with Eddies/Stems?