ROBUST QUERY PROCESSING: Mission Possible!

Jayant Haritsa
Database Systems Lab
Indian Institute of Science, Bangalore, India
Relational DBMS

• Workhorse of today’s Information Industry
  – Commercial
    • IBM DB2, MS SQL Server, Oracle Exadata, HP SQL/MX
  – Public-domain
    • PostgreSQL, MySQL, Berkeley DB

• Extensively researched for over four decades
  – Journals
    • ACM TODS, IEEE TKDE, VLDBJ, …
  – Conferences
    • ACM SIGMOD, IEEE ICDE, VLDB, EDBT, CIKM, …
Design of RDBMS Engines

- **Transaction Processing (ACID)**
  - WAL/ARIES for Atomicity/Recovery
  - 2PL for Concurrency Control

- **Data Access Methods**
  - B-trees/Hashing for Large Ordered Domains
  - Bitmaps for Small Categorical Domains
  - R-trees for Geometric Domains

- **Memory Management**
  - LRU-k (k=2 balances history and responsiveness)

- **Query Processing (SQL)**
  - “Black Art”
Query Execution Plans

• SQL is a declarative language
  – Specifies ends, not means

```sql
select STUDENT.Name, COURSE.Title
from STUDENT, COURSE, REGISTER
where STUDENT.RollNo = REGISTER.RollNo and
    REGISTER.CourseNo = COURSE.CourseNo
```

Unspecified: join order `[((S ⋈ R) ⋈ C) or ((R ⋈ C) ⋈ S) ?]`
join technique [Nested-Loops / Sort-Merge / Hash?]

• DBMS query optimizer identifies the optimal evaluation strategy: “query execution plan”
**Sample Execution Plan**

**Card:**
Output Cardinality (rows)

**Cost:**
Execution Cost (time)
Query Optimization Framework

Declarative Query (Q) → Query Optimizer (Dynamic Programming) → Optimal Plan P(Q)

Operator Execution Cost Estimation Model
- Function of Hardware and DB Engine

Operator Output Cardinality Estimation Model
- Function of Data Distributions and Correlations
Run-time Sub-optimality

The supposedly optimal plan-choice may actually turn out to be highly sub-optimal (e.g. a 1000 times worse!) when the query is executed with this plan. This adverse effect is due to errors in:

(a) cost model
- Reasons: Simple linear models, operator-agnostic features, fixed coefficients, system dynamics ...

(b) cardinality model
- Reasons: Coarse statistics, outdated statistics, attribute value independence (AVI) assumption, multiplicative error propagation, query construction, ...
What have QP folks been doing all these years?

“Elephants”† are highly sensitive animals!

(DB2, Oracle, SQL Server)

(† Stonebraker-speak for enterprise DBMS)
Cardinality Sensitivity Example

### EMPLOYEE
<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cohen</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Giuliani</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Manafort</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Melania</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Ivanka</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Donald</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>Jared</td>
<td>25</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>25</td>
</tr>
<tr>
<td>$10^9$</td>
<td>Eric</td>
<td>25</td>
</tr>
</tbody>
</table>

### MANAGER
<table>
<thead>
<tr>
<th>MID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trump</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>Pence</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>Mnuchin</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>Shanahan</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>Whitaker</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>Bernhardt</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>Perdue</td>
<td>50</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>50</td>
</tr>
<tr>
<td>$10^6$</td>
<td>Ross</td>
<td>50</td>
</tr>
</tbody>
</table>

**EMPLOYEE.AGE $\bowtie$ MANAGER.AGE**

- Output cardinality of the join is **ZERO**
- One new employee aged **50** joins the company
- Output cardinality of the join jumps to a **million**!
- No summary mechanism can capture such **“nanoscopic”** changes
The root of all evil, the Achilles Heel of query optimization, is the estimation of the size of intermediate results, known as cardinalities. The cardinality model can easily introduce errors of many orders of magnitude! With such errors, the wonder isn’t “Why did the optimizer pick a bad plan?” Rather, the wonder is “Why would the optimizer ever pick a decent plan?”
Sound-bites

- **Dave DeWitt**: Query optimizers do terrible job of producing good plans without a lot of hand tuning.
- **Surajit Chaudhuri**: Current state is unsatisfactory with known big gaps in the technology.
- **Little difference between worst-case and average-case in Query Processing**
Prior DB Research (lots!)

• Sophisticated estimation techniques
  – e.g. wavelet histograms, self-tuning histograms, deep learning

• Selection of stable plans
  – e.g. Variance-aware plan selection

• Runtime re-optimization techniques
  – e.g. POP (progressive optimization) [35], RIO (re-optimizer) [6]

Several novel ideas and formulations, but are they robust?
Is there any hope?

Over last decade, several promising advances that collectively promise to soon make robustness a contemporary reality—we survey these techniques in the rest of the tutorial ...
Thanks

Gratefully acknowledge presentation material provided by
- Renata Borovica-Gajic (U of Melbourne, Australia)
- Goetz Graefe (Google Madison Labs, USA)
- Thomas Neumann (TU Munich, Germany)
- Wolf Roediger (Tableau, Germany)
- Wentao Wu (Microsoft Research, USA)
- Srinivas Karthik (IISc Bangalore, India)
- Andreas Kipf (TU Munich, Germany)
- Zongheng Yang (UC Berkeley, USA)
QP Robustness
Importance of Robustness

• Dagstuhl Seminars
  – 2010 (#10381), 2012 (#12321), 2017 (#17222)

• ICDE 2011 panel on Robust Query Processing

• Immediate relevance to database vendors

• Huge impact on database users and customers

• Critical for Big Data world!
ROBUSTNESS DEFINITION

• Multiple perspectives, no consensus
  – If worst-case performance is improved at the expense of average-case performance, is that acceptable?
  – Is it to be defined on a query instance basis, or “in expectation”?
  – …

• Ultimately, robustness definition is application dependent

• Graceful performance profile – no “cliffs”

• Seamless scaling with workload complexity, database size, distributional skew, join correlations

• Provable guarantees on worst-case performance (relative to an offline ideal that makes all the right decisions)
TUTORIAL OUTLINE

• Stage 1: Robust Operators
• Stage 2: Robust Plans
• Stage 3: Robust Query Execution
• Stage 4: Robust Cost Models
• Stage 5: Machine Learning Approaches
• Stage 6: Future Research Directions
Stage 1: Robust Operators
Approaches

• Unified operators
  – Basic Idea: If choice is eliminated, cannot make mistakes, by definition! The key challenge is retaining, in the absence of choice, comparable performance to the multi-choice environment.
  – Smooth Scan (ICDE 2015, VLDBJ 2018 [7])
    • Unifying Table Scan, Index Scan
  – G-join (CS R&D, 2012 [17])
    • Unifying Nested-loops, Sort-merge, Hash-join

• Scaling operators
  – Flow-join (ICDE 2016 [39])
    • Broadcast “heavy hitter” tuples to handle skew in distributed systems
Smooth Scan [7]

(Morph between Sequential Scan and Index Scan)
Sub-optimal Access Paths: Example

**Setting:** TPC-H, SF10, DBMS-COM, Tuned Indexes

Chose index instead of scan due to cardinality underestimation (time: minutes to hours)
Selectivity

Selectivity = Normalized Cardinality [0 to 100%]

\[ \text{sel} = \left( \frac{\text{Output Rows}}{\text{Max Output Rows}} \right) \times 100 \]

\[ \text{COURSE.fees} < 1000 \]

\[ \text{sel} = \left( \frac{4 \times 10^3}{2 \times 10^4} \right) \times 100 \]

\[ = 20 \% \]
Access path selection problem

Tipping Point: One tuple difference in estimation has huge impact on performance

Switching strategies results in performance cliff; cost of change may not be amortized

Selectivity

Execution Time

0 100%

Estimated Actual
Quest for robust access paths

Near-optimal (= min(IS,FS)) throughout entire selectivity range
Smooth Scan in a nutshell

- Statistics-oblivious access path
- Learn result distribution at run-time
- Adapt as you go
**Morphing Mechanism Modes**

1. **Index Access**: Traditional index access
2. **Entire Page Probe**: Index access probes entire page
3. **Gradual Flattening Access**: Probe adjacent region(s)
Morphing policies

- Greedy
- Selectivity Increase Driven
- Elastic

Selectivity increase → Mode Increase
SEL_region ≥ SEL_global
Selectivity decrease → Mode Decrease
SEL_region < SEL_global

Region snooping = Selectivity driven adaptation

X: Page with result
SR: Region selectivity
SG: Global selectivity
## Smooth Scan benefits

<table>
<thead>
<tr>
<th></th>
<th>Index Scan</th>
<th>Full Scan</th>
<th>Sort Scan</th>
<th>Smooth Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid repeated accesses</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fast sequential I/O</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Avoid full table read</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tuples pipelining</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Sort Scan:** Get all qualifying RIDs from the index, sort them, and then sequentially retrieve the records.
TPC-H with Smooth Scan

Setting: TPC-H, SF10, PostgreSQL with Smooth Scan

Robust execution for all queries
Performance Guarantee

Ideal: SortScan without Sorting Cost – i.e. sequentially read only the relevant pages.

\[
\frac{\text{SmoothScan}}{\text{Ideal}} \leq (1 + \frac{\text{rand\_io\_cost}}{\text{seq\_io\_cost}})
\]

For representative HDD parameters, factor is 11, while for SSD, factor is 6.
Limitations

• Several book-keeping data structures required to maintain result semantics (duplicates/ordering)
  – Page ID cache (to not process page twice)
  – Tuple ID Cache (to not produce same tuple twice)
  – Result Cache (for ordered output)
  – Memory Management (for above structures)

• Requires changes to database engine internals
G-join: Generalized Join [17]

(Morph across Indexed-NL Join, Sort-Merge Join, Hybrid-Hash Join)
## Comparative Algorithm Strengths

<table>
<thead>
<tr>
<th></th>
<th>INL Join</th>
<th>SM Join</th>
<th>HH Join</th>
<th>G-join</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted inputs</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Indexed input</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Input size differences</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Basic Idea

- Implement Sort-Merge using concepts from Hash-Join
- If inputs are already sorted, just do Merge Join
- If inputs are not sorted, create internally sorted runs (using replacement selection) as usual for both inputs, but do not carry out merging steps.

Instead, similar to hash partitions, store “key-covering pages” from the small-input ($R$) in a buffer pool, and a single buffer page for the large-input ($S$). Dynamically expand the $R$ buffer pool until it key-covers the buffer page of $S$ – then join the memory-resident pages. After this is done, bring the next $S$ page into memory. Shrink the $R$ buffer pool if any page goes below the key coverage range.
G-join: Phase 1

Phase 1

\[ R_0, R_1, R_2, \ldots, R_K \]

< Cut point, then build

\[ \geq \text{Cut point, then generate runs} \]

Hash table for \( R_0 \)
(M-B pages)

Buffers (B
pages) for run
generation

\[ R_1, R_2, \ldots, R_K \]

\[ S_0, S_1, S_2, \ldots, S_K \]

< Cut point, then probe

Hash table for \( R_0 \)
(M-B pages)

Buffers (B
pages) for run
generation

\[ S_1, S_2, \ldots, S_K \]

\[ \geq \text{Cut point, then generate runs} \]

Figure 3.1 Phase 1 of G-Join
G-join: Phase 2

LK: Low Key. For example, LK_{1,1} represents the Low Key from Run#1 Page #1

HK: High Key. For example, HK_{2,0} represents the High Key from Run#2 Page#0

Figure 3.2 Phase 2 of G-Join
Merge algorithm illustrated

4 pages of 3 runs from input R in the buffer pool

5 runs from input S to be joined one page at a time

Already dropped In buffer pool Yet to be read

Covered key range

Key value domain
Unsorted inputs

![Bar chart showing execution time for different scales and inputs.]

- Execution time [s]
- Scale [number of warehouses]
- Inputs: GJ, HJ, MJ
- Scales: 250, 500, 750, 1000
- Categories: left, right, intermediate, merge, avg deviation

---

September 2020

VLDB Tutorial
Sorted inputs

The diagram shows the execution time for different inputs and scales. The x-axis represents the scale in terms of the number of warehouses, and the y-axis represents execution time in seconds. The bars are color-coded to indicate different stages of processing: left, right, intermediate, and merge. The chart includes data points for various combinations of GJ, HJ, and MJ, with scales of 250, 500, 750, and 1000.
Robust Performance

• **Performance Guarantee:**
  • First-cut theoretical analysis shows rough equivalence to best of existing algorithms

• **Limitations:**
  • Skew in sizes of runs and skew in key value distribution can adversely impact performance

• **Similar “unified” algorithms available for grouping and set operations.** [17]
Stage 2: Robust Plans
APPROACHES

• Least Expected Cost (PODS 99 [11], PODS 02 [12])
  – estimate Distributions instead of Values for parameters

• Cost-Greedy (VLDB 2007 [18])
  – reduce parametric optimal set of plans (POSP) space into low-cardinality (“anorexic”) approximation featuring relatively stable plans

• SEER (VLDB 2008 [19])
  – reduce POSP space into anorexic approximation that can handle arbitrary estimation errors
Cost-Greedy \[18\]
Query Template [Q8 of TPC-H]

Determines how the market share of Brazil in the USA has changed over 1995-1996 for Steel parts

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from ( select YEAR(o_orderdate) as o_year,
           l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation
from part, supplier, lineitem, orders, customer, nation n1, nation n2, region
where p_partkey = l_partkey and s_suppkey = l_suppkey and
  l_orderkey = o_orderkey and o_custkey = c_custkey and
  c_nationkey = n1.n_nationkey and
  r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and
  o_orderdate between '1995-01-01' and '1996-12-31' and
  p_type = 'ECONOMY ANODIZED STEEL'
and s_acctbal ≤ C1 and l_extendedprice ≤ C2
) as all_nations
```
POSGR Plan Diagram

Highly irregular plan boundaries

Intricate Complex Patterns

Extremely fine-grained coverage (P76 ~ 0.01%)

# of plans: 76
Problem Statement

Can the plan diagram be recolored with a smaller set of colors (i.e. some plans are “swallowed” by others), such that

 Guarantee:

No query point in the original diagram has its estimated cost increased, post-swallowing, by more than $\lambda$ percent (user-defined)
CostGreedy

- Optimal plan diagram reduction (w.r.t. minimizing the number of plans/colors) is NP-hard
  - through problem-reduction from classical Set Cover

- CostGreedy is a greedy heuristic-based algorithm with following properties:
  [m is number of query points, n is number of plans in diagram]
  - Time complexity is $O(mn)$
    - linear in number of plans for a given diagram resolution
  - Approximation Factor is $O(\ln m)$
    - bound is both tight and optimal
    - in practice, performance closely approximates offline optimal
Reduced Plan Diagram \[\lambda = 10\%\]

[QT8, OptA*, Res=100]

Complex Plan Diagram

Comparatively smoother contours

Reduced to 5 plans from 76!
Applications of Plan Diagram Reduction

- Quantifies redundancy in plan search space
- Provides better candidates for plan-caching
- Enhances viability of Parametric Query Optimization (PQO) techniques
- Improves efficiency/quality of LEC plans
- Minimizes overheads of multi-plan approaches (e.g. Adaptive Query Processing)

- Identifies selectivity-error resistant plan choices
  - retained plans are robust choices over larger selectivity parameter space
Limitation

Cost Greedy can cause arbitrarily poor performance if the selectivity error is large enough that the actual location of the query falls outside the swallowing region of the estimated location.
Notation

```
select * 
from STUDENT, COURSE, REGISTER 
where S.RollNo = R.RollNo and 
    C.CourseNo = R.CourseNo and 
    C.fees < 1000
```

- $q_e$ – estimated selectivity location in SS (Selectivity Space)
- $q_a$ – actual run-time location in SS
- $P_{oe}$ – optimal plan for $q_e$
- $P_{oa}$ – optimal plan for $q_a$
- $P_{re}$ – replacement plan for $P_{oe}$
Error Locations wrt Plan Replacement Regions

- Endo-optimal
- Swallow
- Exo-optimal

Selectivity X vs Selectivity Y

S

0 100

98% Locations

Inherently robust
SEER [19]
[Selectivity Estimate Error Resistance]
Globally Safe Replacement

• Earlier local constraint:
  \( P_{re} \) can replace \( P_{oe} \) if
  \[ \forall q \in \text{endo-optimality region}, \quad \text{cost}(P_{re}, q) \leq (1 + \lambda) \text{cost}(P_{oe}, q) \]

• New global constraint:
  \( P_{re} \) can replace \( P_{oe} \) only if it guarantees a globally safe space
  \[ \forall q \in \text{selectivity space } S, \quad \text{cost}(P_{re}, q) \leq (1 + \lambda) \text{cost}(P_{oe}, q) \]
Globally Safe Replacement

\[ \text{Safe} \left( P_{re}, P_{oe} \right) \]
Plan Cost Model (2D)

Given selectivity variations $x$ and $y$, for any plan $P$ in the plan diagrams of current optimizers, we can fit:

$$PlanCost_p(x, y) = a_1 x + a_2 y + a_3 xy + a_4 x \log x + a_5 y \log y + a_6 xy \log xy + a_7$$

The specific values of $a_1$ through $a_7$ are a function of $P$. Extension to n-dimensions is straightforward.
Cost Model Fit Example

\[ \text{Original Cost Function} \]

\[ \text{Fitted Cost Function} \]

\[
\text{Cost}(x, y) = 17.9x + 45.9y + 1046xy - 39.5x \log x + 4.5y \log y + 27.6xy \log xy + 97.3
\]
Main Result

Given the 7-coefficient plan cost model, need to perform APC at only the perimeter of the selectivity space to determine global safety.

i.e. Border Safety $\Rightarrow$ Interior Safety!
Limitation

• Although SEER introduces stability into the plan choices, its performance guarantees are with respect to $P_{oe}$, the optimal plan at the estimated location (i.e. the native optimizer’s plan).

• Ideally, we would like performance guarantees to be with respect to $P_{oa}$, the optimal plan at the actual location (i.e. the “God’s plan”).
Stage 3: Robust Execution
Performance Metrics

- $q_e$ – estimated selectivity location in SS
- $q_a$ – actual run-time location in SS
- $P_{oe}$ – optimal plan for $q_e$
- $P_{oa}$ – optimal plan for $q_a$

$SubOpt(q_e, q_a) = \frac{\text{cost}(P_{oe}, q_a)}{\text{cost}(P_{oa}, q_a)} \in [1, \infty)$

$MaxSubOpt (MSO) = \text{MAX}[SubOpt(q_e, q_a)] \ \forall q_e, q_a \in SS$

Note: Metric is now with respect to the ideal plan
APPROACHES

• Bounded Impact (PVLDB 2009 [36])
  – performance guarantee with quartic dependency on error magnitude

• Plan Bouquet (SIGMOD 14 / TODS 16 [14])
  – discovery-based approach to selectivities
  – error-independent guarantees with linear dependency on plan diagram density

• Spill-Bound (ICDE 16 / TKDE 19 [25])
  – platform-independent guarantee with quadratic dependency on error dimensionality

• Frugal Spill-Bound (PVLDB 2018 [26])
  – extension to ad-hoc queries
Measuring Cardinality Estimation Errors

Popular error metrics (= optimization goals)

\[ l_2 = \sqrt{(f_e(x) - f_a(x))^2} \]

\[ l_\infty = \max |(f_e(x) - f_a(x)| \]

Minimizing these error metrics can lead to arbitrarily bad plans!
**Q(quotient) Error**

- Errors propagate multiplicatively, so metric should also be multiplicative
- It should be symmetric wrt over- and under-estimation

\[ l_q = \max_{x} \frac{\max(f_e(x), f_a(x))}{\min(f_e(x), f_a(x))} \]

- actual cardinality 10, estimation 100 \( \Rightarrow l_q = 10 \)
- actual cardinality 10, estimation 1 \( \Rightarrow l_q = 10 \)

Knowing \( q \)-error provides **bounds on resulting plan performance!**
Cost Bounds Implied by Q-error

• Theorem:
  Let all joins be Sort-Merge or all be Grace-Hash. Then

  \[ MSO \leq q^4 \]

  where \( q \) is the maximum q-error taken over all intermediate results.

Problems: \( q \) can be arbitrarily large
  \( q \) is usually not known in advance
Plan Bouquet [14]
Approach

- Plan Bouquet is a new query processing technique, that completely abandons estimating operator selectivities.
- Instead, run-time selectivity discovery using compile-time selected bouquet of plans.
  - Provides worst case performance guarantees wrt ideal that magically knows the correct selectivities. e.g. for single error-prone selectivity, relative guarantee of 4.
  - Empirical performance well within guaranteed bounds on industrial-strength environments.
Basic Assumption

- Plan Cost Monotonicity (PCM)

For any plan \( P \) and distinct locations \( q_1 \) and \( q_2 \)

\[
\text{Cost}(P, q_1) < \text{Cost}(P, q_2) \quad \text{if} \quad q_1 < q_2
\]

(i.e. spatial domination \( \Rightarrow \) cost domination)
Contemporary Optimizer Behavior on 1D Selectivity Space
Parametric Optimal Set of Plans (POSP)

(Parametric version of Example Query)

```
select *
from STUDENT, COURSE, REGISTER
where S.RollNo = R.RollNo and
  C.CourseNo = R.CourseNo and
  C.fees < $1
```
POSP Performance Profile (across SS)
Sub-optimality Profile (across SS)

- SubOpt \((q_e = 1\%, q_s = 99\%)\) = 20
- SubOpt \((q_e = 80\%, q_s = 0.01\%)\) = 100
- MaxSubOpt = 100

Estimated Costs (log-scale)

Selectivity COURSE (log-scale)

September 2020

VLDB Tutorial
Plan Bouquet Behavior on 1D Selectivity Space
Bouquet Identification

Step 1: Draw cost steps with cost-ratio $r=2$ (geometric progression).

Step 2: Find plans at intersection of optimal profile with cost steps

Bouquet = \{P1, P2, P3, P5\}
Let \( q_a = 5\% \)

1. Execute P1 with budget IC1(1.2E4) 
   *Throw away results of P1*

2. Execute P1 with budget IC2(2.4E4) 
   *Throw away results of P1*

3. Execute P1 with budget IC3(4.8E4) 
   *Throw away results of P1*

4. Execute P1 with budget IC2(9.6E4) 
   *Throw away results of P1*

5. Execute P2 with budget IC5(1.9E5) 
   *Throw away results of P1*

6. Execute P3 with budget IC6(3.8E5) 
P3 completes with cost 3.4E5
Let $q_a = 5\%$

Bouquet Cost = $3.4 \times 10^5$ (P3) + $1.92 \times 10^5$ (P2) + $0.96 \times 10^5$ (P1) + $0.48 \times 10^5$ (P1) + $0.24 \times 10^5$ (P1) + $0.12 \times 10^5$ (P1) = $7.1 \times 10^5$

SubOpt (*, 5%) = $7.1/3.4 = 2.1$

With obvious optimization
SubOpt(*, 5%) = $6.3/3.4 = 1.8$

Bouquet Execution
Bouquet Performance (EQ)

Native Optimizer
MaxSubOpt = 100

Bouquet
MaxSubOpt = 3.1
Worst Case Cost Analysis

P_k would complete within its budget when \( q_a \in (q_{k-1}, q_k] \)
1D Performance Bound

\[
C_{\text{bouquet}}(q_{k-1}, q_k] = \text{cost}(IC_1) + \text{cost}(IC_2) + \ldots + \text{cost}(IC_{k-1}) + \text{cost}(IC_k) \\
= a + ar + \ldots + ar^{k-2} + ar^{k-1} \\
= \frac{a(r^k - 1)}{(r - 1)}
\]

\[
C_{\text{optimal}}(q_{k-1}, q_k] \geq ar^{k-2} \quad \text{(Implication of PCM)}
\]

\[
\text{SubOpt}_{\text{bouquet}}(*, q_a) \leq \frac{1}{ar^{k-2}} \times \frac{a(r^k - 1)}{(r - 1)} \leq \frac{r^2}{r - 1} \quad \forall q_a \in (q_{k-1}, q_k]
\]

Best performance achievable by any deterministic online algorithm!

Reaches minima at \( r = 2 \)

\[\Rightarrow \text{MSO} = 4\]
Bouquet Approach in 2D SS
2D Bouquet Identification

Cost (normalized)

Plans

Isocost Planes

Multiple Plans per contour

sel-X

sel-Y

September 2020

VLDB Tutorial
Characteristics of 2D Contours

2D contours
- Hyperbolic curves
- Multiple plans per contour

Third quadrant coverage (due to PCM)
$P^k_2$ can complete any query with actual selectivity $(q_a)$ in the shaded region within $\text{cost(IC}_k)$
Crossing 2D Contours

Covered by all plans in contour

Covered by only one plan in contour

⇒ Entire set of contour plans must be executed to fully cover all locations under IC_k
2D Performance Analysis

- When $q_a \in (IC_{k-1}, IC_k]$

\[
C_{bouquet}(q_a) = \sum_{i=1}^{k} [n_i \times \text{cost}(IC_i)]
\]

$\rho = \max(n_i)$

\[
C_{bouquet}(q_a) \leq \rho \times \sum_{i=1}^{k} \text{cost}(IC_i)
\]

$SubOpt_{bouquet}(q_a) = 4\rho$ (Using 1D Analysis)

- MSO = $4\rho$

Bound for N-dimensions: $MSO = 4 \times \rho_{IC\text{surface}}$
Dealing with large $\rho$

- In practice, $\rho$ can often be large, even in 100s, making the performance guarantee of $4\rho$ impractically weak

- Reducing $\rho$:
  
  Anorexic POSP reduction
  (from CostGreedy)
MSO guarantees (compile time)

<table>
<thead>
<tr>
<th>Query (dim)</th>
<th>MSO Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q5 (3D)</td>
<td>14.4</td>
</tr>
<tr>
<td>Q7 (3D)</td>
<td>14.4</td>
</tr>
<tr>
<td>Q8 (4D)</td>
<td>33.6</td>
</tr>
<tr>
<td>Q7 (5D)</td>
<td>43.2</td>
</tr>
<tr>
<td>Q15 (3D)</td>
<td>14.4</td>
</tr>
<tr>
<td>Q96 (3D)</td>
<td>14.4</td>
</tr>
<tr>
<td>Q7 (4D)</td>
<td>19.2</td>
</tr>
<tr>
<td>Q19 (5D)</td>
<td>38.4</td>
</tr>
<tr>
<td>Q26 (4D)</td>
<td>24.0</td>
</tr>
<tr>
<td>Q91 (4D)</td>
<td>43.2</td>
</tr>
</tbody>
</table>
Empirical Evaluation
Experimental Testbed

- Database Systems: PostgreSQL and COM (commercial engine)
- Databases: TPC-H and TPC-DS (standard benchmarks)
- Physical Schema: Indexes on all attributes present in query predicates

- Workload: 10 complex queries from TPC-H and TPC-DS
  - with SS having upto 5 error dimensions (join-selectivities)

- Metrics: Computed MSO using Abstract Plan Costing over SS
Performance on PostgreSQL

Native Optimizer

Plan Bouquet

Log-scale

MSO

MSO bounds
Performance with Commercial System

![Bar chart showing performance comparison between NAT and BOU in a commercial system. The x-axis represents different configurations (3D_H_Q5b and 4D_H_Q8b), and the y-axis shows performance metrics (1.0E+00, 1.0E+02, 1.0E+04). The chart illustrates higher performance for BOU compared to NAT.]
Summary

- Plan bouquet approach achieves
  - bounded performance sub-optimality
    - using a (cost-limited) plan execution sequence guided by isocost contours defined over the optimal performance curve
  - robust to changes in data distribution
    - only $q_a$ changes – bouquet remains same
  - easy to deploy
    - bouquet layer on top of the database engine
  - repeatability in execution strategy (important for industry)
    - $q_e$ is always zero, depends only on $q_a$
    - independent of metadata contents
Limitations of PlanBouquet

- Enormous offline computational effort to produce the plan diagram, suitable only for “canned” queries
  - Partially addressed by enumerating only the contours, not the entire diagram
- Practical guarantee values are predicated on anorexic reduction holding true
- Guarantee of $4\rho$ depends on plan diagram complexity, making it not portable across query optimizers, databases and hardware systems
FOLLOW-UP WORK

• Spill-Bound (ICDE 16 / TKDE 19 [25])
  – Half-space pruning instead of hypograph pruning
  – \( \text{MSO} = D^2 + 3D \) (where \( D \) is dimensionality of SS)
  – platform-independent guarantee
  – Lower bound of \( \Omega(D) \)

• Frugal Spill-Bound (PVLDB 2018 [26])
  – extension to ad-hoc queries
  – exponential decrease in overheads for linear relaxation in MSO guarantee
Stage 4: Robust Cost Models
Approaches

- Learning-based approaches (ICDE 2009, ICDE 2012 [4], PVLDB 2019 [40])

- Statistical approaches (ICDE 2013 [45], PVLDB 2013 [44], PVLDB 2014 [47])
Optimizer’s Cost Estimates: Unusable

Direct Scaling:
Predict the execution time $T$ by scaling the cost estimate $C$, i.e., $T = a \cdot C$

Fig. 5 of [4]
Why Does Direct Scaling Fail?

- PostgreSQL’s cost model

\[ C = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o \]

Scaling

\[ T = a \cdot C = c'_s \cdot \left( n_s + n_r \frac{c_r}{c_s} + n_t \frac{c_t}{c_s} + n_i \frac{c_i}{c_s} + n_o \frac{c_o}{c_s} \right) \]

\[ c'_s = a \cdot c_s = a \cdot 1.0 = a \]

- Assumptions for scaling fail in practice
  - *Ratios* between the \( c \) values are incorrect.
  - \( n \) values are incorrect.

- Solution: Proper calibration

<table>
<thead>
<tr>
<th>Cost Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_s ): seq_page_cost</td>
<td>1.0</td>
</tr>
<tr>
<td>( c_r ): rand_page_cost</td>
<td>4.0</td>
</tr>
<tr>
<td>( c_t ): cpu_tuple_cost</td>
<td>0.01</td>
</tr>
<tr>
<td>( c_i ): cpu_index_tuple_cost</td>
<td>0.005</td>
</tr>
<tr>
<td>( c_o ): cpu_operator_cost</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

September 2020

VLDB Tutorial
Calibrated $c$ and $n$

- Cost models become much more effective.

Prediction by Scaling:
$$T_{pred} = a \cdot (\sum c \cdot n)$$

Prediction by Calibration:
$$T_{pred} = \sum c' \cdot n'$$
Main Idea

- Calibrate $c$: use profiling queries
- Calibrate $n$: refine cardinality estimates
Profiling Queries For PostgreSQL

*Isolate* the unknowns and solve them *one per equation*

- **q₁**: select * from R
  - R in memory
  - $t₁ = cᵗ \cdot nᵗ₁$

- **q₂**: select count(*) from R
  - R in memory
  - $t₂ = cᵗ \cdot nᵗ₂ + cₒ \cdot nₒ₂$

- **q₃**: select * from R where R.A < a
  - (R.A with an index)
  - R in memory
  - $t₃ = cᵗ \cdot nᵗ₃ + cᵢ \cdot nᵢ₃ + cₒ \cdot nₒ₃$

- **q₄**: select * from R
  - R on disk
  - $t₄ = cₛ \cdot nₛ₄ + cᵗ \cdot nᵗ₄$

- **q₅**: select * from R where R.B < b
  - (R.B *unclustered* index)
  - R on disk
  - $t₅ = cₛ \cdot nₛ₅ + cᵣ \cdot nᵣ₅ + cᵗ \cdot nᵗ₅ + cᵢ \cdot nᵢ₅ + cₒ \cdot nₒ₅$
Calibrating the *n* values

- The *n* values are *functions* of *N* values (i.e., input cardinalities).
  - Calibrating the *n* values ⇒ Calibrating the *N* values

**Example 1** (In-Memory Sort)

\[
sc = \left(2 \cdot N_t \cdot \log N_t \right) \cdot c_o + tc \ of \ child
\]

\[
rc = c_t \cdot N_t
\]

**Example 2** (Nested-Loop Join)

\[
sc = sc \ of \ outer \ child + sc \ of \ inner \ child
\]

\[
rc = c_t \cdot N_t^o \cdot N_t^l + N_t^o \cdot rc \ of \ inner \ child
\]

*sc*: start-cost  \  *rc*: run-cost  \  *tc* = *sc* + *rc*: total-cost

*NT*: # of input tuples
Refine Cardinality Estimates

- Different perspective than the norm (query optimization)

<table>
<thead>
<tr>
<th></th>
<th>Query Optimization</th>
<th>Execution Time Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Plans</td>
<td>Hundreds/Thousands</td>
<td>1</td>
</tr>
<tr>
<td>Time per Plan</td>
<td>Must be very short</td>
<td>Can be a bit <em>longer</em></td>
</tr>
<tr>
<td>Precision</td>
<td>Important</td>
<td><em>Critical</em></td>
</tr>
<tr>
<td>Approach</td>
<td>Histograms (dominant)</td>
<td><em>Sampling</em> (one option)</td>
</tr>
</tbody>
</table>
A Sampling-Based Estimator

- Estimate the selectivity $\rho_q$ of a select-join query $q$.

$q : R_1 \bowtie R_2$

Partition

$n$ samples (w/ replacement)

The estimator $\hat{\rho}_q$ is unbiased and strongly consistent

\[ \rho_1 = \frac{|B_{11} \bowtie B_{22}|}{|B_{11}| \times |B_{22}|} \]

\[ \hat{\rho}_q = \frac{1}{n} \sum_{i=1}^{n} \rho_i \]
Cardinality Refinement Algorithm

- Design the refinement algorithm based on the previous sampling formula.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The estimator needs random I/Os at runtime to take samples.</td>
<td>Take samples offline and store them as tables in the database.</td>
</tr>
<tr>
<td>Query plans usually contain more than one operator.</td>
<td>Estimate multiple operators in a single run, by reusing partial results.</td>
</tr>
<tr>
<td>The estimator only works for select/join operators.</td>
<td>Rely on PostgreSQL’s cost models for aggregates.</td>
</tr>
</tbody>
</table>
Cardinality Refinement Algorithm (Example)

Plan for q:

\[ q_1 = R_1 \bowtie R_2 \]
\[ q_2 = R_1 \bowtie R_2 \bowtie R_3 \]

For \textit{agg}, use PostgreSQL’s estimates based on the \textit{refined} input estimates from \( q_2 \).

\[ R_1^s, R_2^s, R_3^s \text{ are samples (as tables) of } R_1, R_2, R_3 \]

\[ \hat{\rho}_{q_1} = \frac{|R_1^s \bowtie R_2^s|}{|R_1^s| \times |R_2^s|} \]
\[ \hat{\rho}_{q_2} = \frac{|R_1^s \bowtie R_2^s \bowtie R_3^s|}{|R_1^s| \times |R_2^s| \times |R_3^s|} \]
Experimental Settings

• PostgreSQL 9.0.4, Linux 2.6.18

• TPC-H 1GB and 10GB databases
  – Both uniform and skewed data distribution

• Two different hardware configurations
  – PC1: 1-core 2.27 GHz Intel CPU, 2GB memory
  – PC2: 8-core 2.40 GHz Intel CPU, 16GB memory
## Calibrating Cost Units

**PC1:**

<table>
<thead>
<tr>
<th>Cost Unit</th>
<th>Calibrated (ms)</th>
<th>Calibrated (normalized to $c_s$)</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_s$: seq_page_cost</td>
<td>5.53e-2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$c_r$: rand_page_cost</td>
<td>6.50e-2</td>
<td>1.2</td>
<td>4.0</td>
</tr>
<tr>
<td>$c_t$: cpu_tuple_cost</td>
<td>1.67e-4</td>
<td>0.003</td>
<td>0.01</td>
</tr>
<tr>
<td>$c_i$: cpu_index_tuple_cost</td>
<td>3.41e-5</td>
<td>0.0006</td>
<td>0.005</td>
</tr>
<tr>
<td>$c_o$: cpu_operator_cost</td>
<td>1.12e-4</td>
<td>0.002</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

**PC2:**

<table>
<thead>
<tr>
<th>Cost Unit</th>
<th>Calibrated (ms)</th>
<th>Calibrated (normalized to $c_s$)</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_s$: seq_page_cost</td>
<td>5.03e-2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$c_r$: rand_page_cost</td>
<td>4.89e-1</td>
<td>9.7</td>
<td>4.0</td>
</tr>
<tr>
<td>$c_t$: cpu_tuple_cost</td>
<td>1.41e-4</td>
<td>0.0028</td>
<td>0.01</td>
</tr>
<tr>
<td>$c_i$: cpu_index_tuple_cost</td>
<td>3.34e-5</td>
<td>0.00066</td>
<td>0.005</td>
</tr>
<tr>
<td>$c_o$: cpu_operator_cost</td>
<td>7.10e-5</td>
<td>0.0014</td>
<td>0.0025</td>
</tr>
</tbody>
</table>
Prediction Precision

• Metric of precision
  – Mean Relative Error (MRE)
  – (questionable as compared to q-error)

\[ \frac{1}{M} \sum_{i=1}^{M} \frac{|T_{i}^{pred} - T_{i}^{act}|}{T_{i}^{act}} \]

• Dynamic database workloads
  – Unseen queries frequently occur

• Compare with existing approaches
  – Direct scaling
  – Machine learning approaches
Precision on TPC-H 1GB DB

Uniform data:

- $E_t$: c’s (calibrated) + n’s (true cardinalities)
- $E_o$: c’s (calibrated) + n’s (cardinalities by optimizer)
- $E_s$: c’s (calibrated) + n’s (cardinalities by sampling)
Precision on TPC-H 1GB DB (Skewed)

Skewed data:

\[ E_t: c's \text{ (calibrated)} + n's \text{ (true cardinalities)} \]
\[ E_o: c's \text{ (calibrated)} + n's \text{ (cardinalities by \textit{optimizer})} \]
\[ E_s: c's \text{ (calibrated)} + n's \text{ (cardinalities by \textit{sampling})} \]
Summary

- Systematic framework to calibrate the cost units and refine the cardinality estimates used by current cost models.

- Showed that current statistical cost models are quite effective in query execution time prediction after proper calibration, and the additional overhead is affordable in practice.
Stage 5: ML Approaches
Motivation

• Over the past three years, a flood of publications [16, 22, 23, 27, 28, 34, 37, 40, 48, 49, 50, 51, 52, 53, 54, 56, 57, 59, … !] advocating deep-learning-based approaches for both cardinality-estimation and cost-estimation.

• Basic idea is to replace coarse parametrized models with fine-grained learnt models. The expectation is that these deep models are better able to capture the in situ data and system behavior due to their flexibility, scalability and lack of prior assumptions.
Approaches

• Two broad classes
  – query-based (supervised learning)
    • Models constructed by training on a large set of queries and leveraging the observed values during execution as labels
  – data-based (unsupervised learning)
    • Model the joint probability density functions of the underlying data to capture distributions and correlations
MSCN [28]
(Multi-set Convolutional Neural Network)
Framework

- Estimating cardinalities for correlated joins, since they are especially hard to model well
  - e.g. French actors are more likely to participate in romantic movies than actors of other nationalities

- Key Ideas
  - Set-based model (based on DeepSets):
    \((A \bowtie B) \bowtie C\) and \(A \bowtie (B \bowtie C)\) are both represented as \{A, B, C\}
  - Integrates sampling:
    use bitmaps of qualifying base table samples as ML features

- Advantages
  - Learns join-crossing correlations
  - Addresses “0-tuple” situations: model relies on query features in cases when no or very few samples qualify
1) Obtaining Training Data

- Generate synthetic queries using schema information (data types and constraints) and the actual values from the database
- Execute queries on a snapshot of the database to obtain true cardinalities
- Annotate queries with bitmaps indicating qualifying base table samples
2) Feature Selection and Representation

- Query features (tables, predicates, joins, ...) are one-hot encoded
- Values (literals) and true cardinalities are normalized to $[0,1]$

```sql
SELECT * FROM title t, movie_companies mc WHERE t.id = mc.movie_id
AND t.production_year > 2010 AND mc.company_id = 5
```

Table set $\begin{bmatrix} 0 & 1 & 0 & 1 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 1 \end{bmatrix}$

Join set $\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$

Predicate set $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0.72 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.14 \end{bmatrix}$

- True cardinality (label): 665 (encoded as 0.1 if max = 6650)
3) Set-based ML model

- Four fully-connected multi-layer neural networks (MLPs)
4) Optimization Metric

- **q-error**: multiplicative ratio between true and estimated cardinalities.

- **Goal**: minimize mean q-error over the training set
Estimation Quality

- IMDB data-set: contains many correlations
- Synthetic queries: only equality and range predicates
Generalizing to More Joins

![Box plots comparing the under. [log scale] over. for PostgreSQL and MSCN across different numbers of joins.](image)
Training Convergence

![Graph showing training convergence](image)
Summary

• Deep learning can capture complex correlations and address limitations of pure sampling when there is a good match between the training and testing environments
NARU [48]
(Neural Relation Understanding)
Learning Model

Training
Learn joint data distribution with deep autoregressive model

Inference
Monte Carlo integration to answer range density queries

Point query

Range query

Data Source

Encode

Likelihood Model

Decode

salaries

age

salary=10k
age=30

5k<=salary<=15k
20<=age<=28

unsupervised loss
(maximum likelihood)
### Joint Distribution

#### Data (strings, nums, dates, ...)

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2000</td>
</tr>
<tr>
<td>25</td>
<td>10,000</td>
</tr>
<tr>
<td>24</td>
<td>2000</td>
</tr>
<tr>
<td>24</td>
<td>2000</td>
</tr>
</tbody>
</table>

#### Joint Distribution (P)

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
<th>P(A,S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2000</td>
<td>1/4</td>
</tr>
<tr>
<td>25</td>
<td>10,000</td>
<td>1/4</td>
</tr>
<tr>
<td>24</td>
<td>2000</td>
<td>2/4</td>
</tr>
</tbody>
</table>

- **Everything else**
  - 0

#### %rows matched by the query?

\[
\text{SELECT } * \text{ FROM } T \text{ WHERE Age } \leq 25 \text{ AND Salary } \leq 2000
\]

**selectivity(Q) ≡ density(Q):**

\[
\text{density}(\text{Age} \leq 25 \text{ && Salary} \leq 2000)
\]

Integrating the joint yields density(Q):

- **Valid Age:** [24, 25]; **Valid Salary:** [2000].
- Sum up the densities from valid points.

\[
= 1/4 + 2/4 = 0.75
\]
Learning the Joint Distribution

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
<th>$P(A,S)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2000</td>
<td>1/4</td>
</tr>
<tr>
<td>25</td>
<td>10,000</td>
<td>1/4</td>
</tr>
<tr>
<td>24</td>
<td>2000</td>
<td>2/4</td>
</tr>
<tr>
<td></td>
<td>everything else</td>
<td>0</td>
</tr>
</tbody>
</table>

Use a **deep autoregressive model** to learn:

$$P(x) = \prod_{i=1}^{n} P(x_i|x_{<i})$$

where $x$ is an $n$-dimensional tuple.

Calculation becomes

$$\text{density}(\text{Age} \leq 25 \land \text{Salary} \leq 2000)$$

$$\approx \text{Model}(25, 2000) + \text{Model}(24, 2000)$$
Learning the Joint

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
<th>( P(A,S) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2000</td>
<td>1/4</td>
</tr>
<tr>
<td>25</td>
<td>10,000</td>
<td>1/4</td>
</tr>
<tr>
<td>24</td>
<td>2000</td>
<td>2/4</td>
</tr>
<tr>
<td></td>
<td>everything else</td>
<td>0</td>
</tr>
</tbody>
</table>

Joint Distribution (\( P \))

Use a deep autoregressive model to learn:

\[
P(x) = \prod_{i=1}^{n} P(x_i | x_{<i})
\]

where \( x \) is an \( n \)-dimensional tuple.

Comparing to previous work:

- Chain-rule factorization means no information loss
- Independence assumption loses information
  - 1D Histogram: \( P(A,B,C) \approx P(A) P(B) P(C) \)
  - Partial Independence: \( P(A,B,C) \approx P(A,B) P(C) \)
  - Conditional Independence: \( P(A,B,C) \approx P(B) P(C|B) P(A|C) \)

Not materialized; Emitted on-demand by model
Model Training

**Output**: probability distributions over columns

```
25 10K
24 2K
25 2K
... ...
```

**Input**: each tuple

Plug in any deep autoregressive model:

- Masked MLP
- MADE [ICML’15]
- ResMADE [Nash et al. ’19]
- Transformer [NIPS’17] and variants
- WaveNet
- ...

Train via Maximum Likelihood
Range Density Estimates

DAR model outputs point density. Require range density at inference time.

density(Age\leq 25 \&\& Salary\leq 2000)
2 valid points to forward pass

density(X_1 \text{ in } R_1, \ldots, X_n \text{ in } R_n)
has |R_1| \times |R_2| \times \ldots \times |R_n| points (exponential)

Insight: not all points in the queried region are meaningful
\rightarrow use \textbf{Monte Carlo integration} (sampling) to approximate range density
Approximate Inference

\[ 5k \leq \text{salary} \leq 15k \]
\[ 20 \leq \text{age} \leq 28 \]

\[ \text{P(Age in [20...28] } \& \& \text{ Salary in [5k...15k])} \]

Exact inference: exponential in #columns of the table

Progressive Sampling

Sample X1
\[ \sim P(X_1 | X_1 \text{ in R1}) \]

Sample X2
\[ \sim P(X_2 | X_2 \text{ in R2, x1_sample}) \]

Weight densities appropriately

Use sample from each dimension to progressively zoom into high-mass region
Progressive Sampling
Estimation Accuracy

• Supervised: Few hours to collect 10K training queries
• Unsupervised: Few minutes to read and train

Dataset DMV (11M tuples, 11 columns)
Workload 5-11 range/eq filters; 2K queries
Model Masked MLP (#params: 3M; ~1% data size)
Limitations of Learning approaches

- **Universality**
  - Ability to handle unseen adhoc queries is suspect
- **Explainability**
  - Do not provide an intuitive confirmation of the approach
- **Guarantees**
  - Average case may be excellent, but worst-case can be arbitrarily poor
- **Heavy-weight**
  - May require expensive training phase
- **Uncertainty estimation**
  - Hard to quantify the risk involved in trusting the model

**Open Problem:**
Compare Algorithmic (Algebra+Geometry) vs Function-fitting approaches
Putting it all together
Good news

The proposed techniques are complementary and can work together!
New RQP Architecture: Plan-level

- SpillBound for Performance Guarantees
- CostGreedy for Anorexic Plan Density
- Q-Calibrated + Naru-Sampled Cost Model for Contours
New RQP Architecture: Intra-Plan

G-Join, FlowJoin for Data Processing

SmoothScan for Data Access
Stage 6: Future Research
1) Structure of Query Graphs

- Graph structure (chain, star, cycle, etc.) has significant impact on robustness guarantees
  - Tighter guarantees for chain \((8D - 6)\) as compared to star \((D^2 + 3D)\)

Open Problem: MSO derivations based on query graph type
2) Refined Cost Model Calibration

- Calibration discussed previously assumed the Postgres basic 5-parameter model as a given for the entire suite of operators.

- **Open Problem:** Add operator-specific features and operator-specific calibration of the coefficients, and see if accuracy can be improved.
3) Robustness Benchmarks

- Standard industry benchmarks (e.g. TPC-DS) are oriented towards performance, not robustness.

- Recent proposals on benchmarks:
  - Optimizer Benchmark (OptMark) (CIKM 16 [28])
    - TPC-DS synthetic data, examines plan coverage and estimation of plans better than optimizer’s choice; does not cover magnitude of cost differences
  - Join-Order Benchmark (JOB) (VLDBJ 18 [26])
    - Based on IMDB real data with heavy skew and correlation, and join-heavy queries, q-error
  - Optimizer Torture Test (OTT) (SIGMOD 16 [43])
    - Two-column relations, one join attribute and one selection, the two columns are highly correlated (in fact, identical values!)

- Open Problem: Design non-pathological realistic benchmarks that highlight robustness issues (e.g. performance cliffs)
END RQP TUTORIAL
REFERENCES

REFERENCES (contd)


REFERENCES (contd)


Additional References


