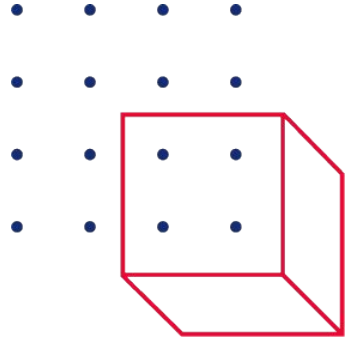
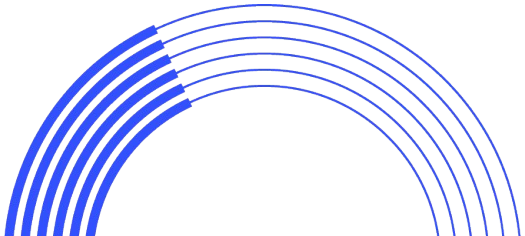
PingCAP



Introduction to Optimizer Evaluation

Yuanjia Zhang



Overview

PingCAP

1. Why is it difficult to test optimizers
2. Some methods to test an optimizer
    a. Analyzing Plan Diagrams of Database Query Optimizers [Diagram]
    b. Exact Cardinality Query Optimization for Optimizer Testing [ECQO]
    c. Testing Cardinality Estimation Models in SQL Server [CE Testing]
    d. Testing the Accuracy of Query Optimizers [TAQO]
    e. How Good Are Query Optimizers, Really? [GQO]
3. Summary

Why is it difficult to test optimizers

PingCAP

1. its inherent statistical nature
2. the infinite input space and plan space
3. components are highly coupled
    a. inside: stats, CE, cost, rule
    b. outside: adaptive execution, self-tuing(feedback?)
4. lack of user workloads (for TiDB)
5. ...

Diagram: Introduction

Link: Analyzing Plan Diagrams of Database Query Optimizers

Introduction: introduce a new method(tool) to analyze plan choices of an optimizer, and find some interesting insights of current optimizers.

The main assumption: the optimal plan choice is **primarily** a function of the sectivities of the base relations.

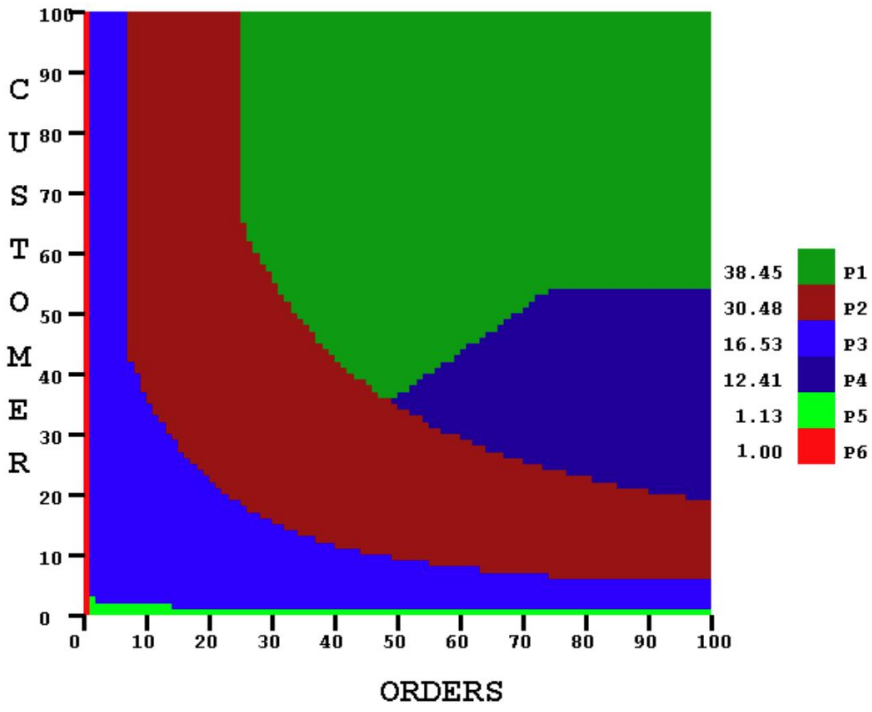
An example:

select * from t1, t2 where

t1.id=t2.id and **cond(t1)** and **cond(t2)**

Some limitations:

1. queries with 3 or more tables?
2. data distribution is not considered?

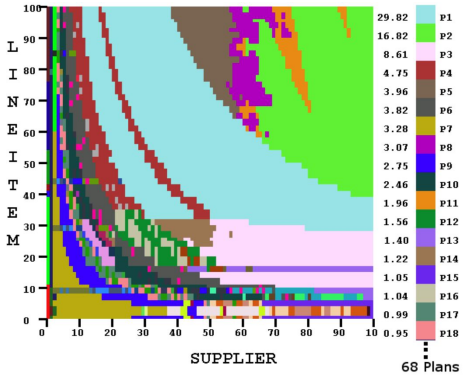


(a) **Plan Diagram**

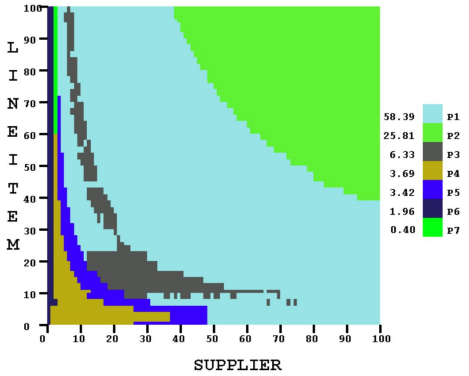
Diagram: Fine-grained Choices

1. current optimizers may be over-sophisticated
2. ⅔ of the plans are liable to be eliminated through a simple algorithm (without increasing the cost by more than 10%)
3. in another paper, Identifying Robust Plans through Plan Diagram Reduction:
   a. small-sized plans are not robust
   b. most small-sized plans can be eliminated within a little cost increase

TPC-H Query Number	OptA Plan Card	OptA 80% Coverage	OptA Gini Index	OptB Plan Card	OptB 80% Coverage	OptB Gini Index	OptC Plan Card	OptC 80% Coverage	OptC Gini Index
2	22	18%	0.76	14	21%	0.72	35	20%	0.77
5	21	19%	0.81	14	21%	0.74	18	17%	0.81
7	13	23%	0.73	6	50%	0.46	19	15%	0.79
8	31	16%	0.81	25	25%	0.72	38	18%	0.79
9	63	9%	0.88	44	27%	0.70	41	12%	0.83
10	24	16%	0.78	9	22%	0.69	8	25%	0.75
18	5	60%	0.33	13	38%	0.57	5	20%	0.75
21	27	22%	0.74	6	17%	0.80	22	18%	0.81
Avg(dense)	**28.7**	**17%**	**0.79**	**24.5**	**23%**	**0.72**	**28.8**	**16%**	**0.8**



(a) **Complex Plan Diagram**



(b) **Reduced Plan Diagram**

Diagram: Duplicates and Islands

Figure 5: plan islands, hash-join v.s. nested-loop join

Figure 6: plan duplicates, nested-loop join v.s. sort-merge join

Reason: Caused by two or more competing plans have close costs in that area.

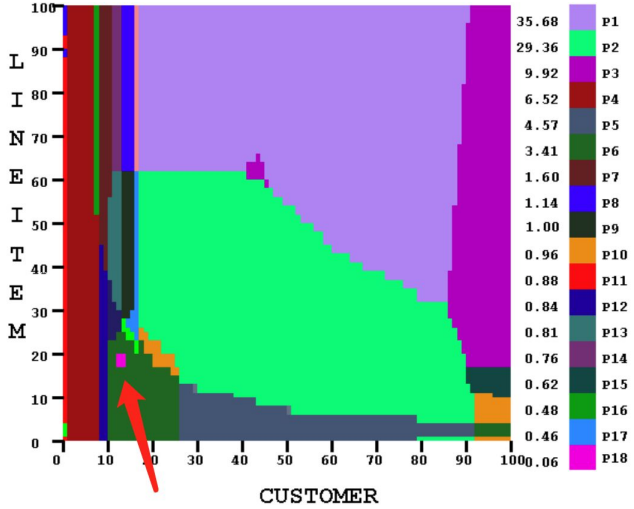


Figure 5: **Duplicates and Islands (Query 10, OptA)**

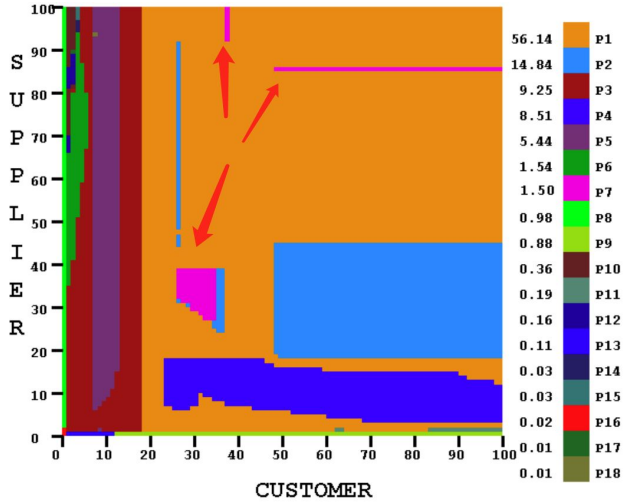


Figure 6: **Duplicates and Islands (Query 5, OptC)**

Diagram: Switch-Point

Figure 7, 8: plan switch-point, join order

Reason: cost-models are perhaps discretized

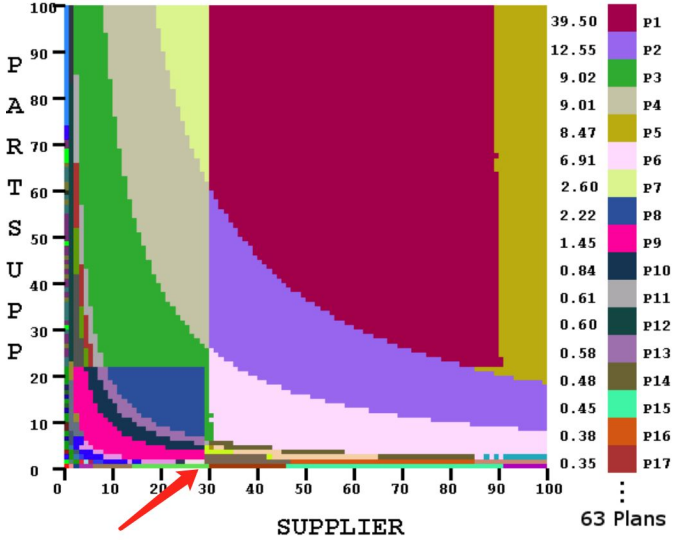


Figure 7: **Plan Switch-Point (Query 9, OptA)**

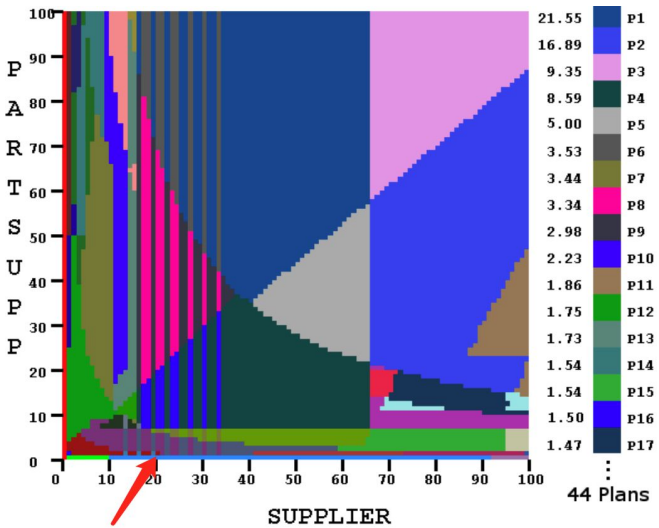


Figure 8: **Venetian Blinds Pattern (Query 9, OptB)**

Diagram: Speckle Pattern

This is a realistic practice in SQL Sever Team when refactoring the CE Model.

Snowy Plan Diagram(Figure 7) is caused by floating point inconsistent in the code

This reduction in distinct plans will manifest itself in less volatile query plan selection

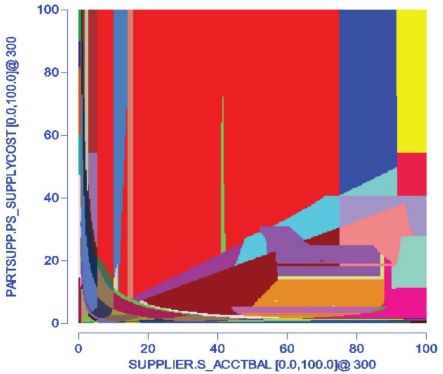


Figure 6: Plan diagram of TPC-H, Query 9, for the old cardinality estimation model. Number of distinct plans: 78.

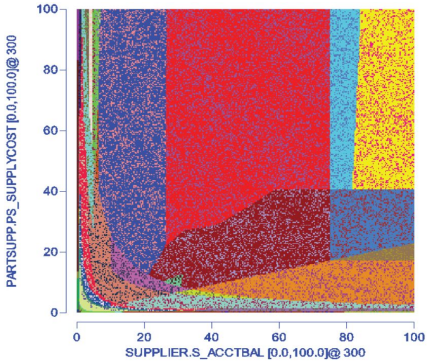


Figure 7: Original (Snowy) Plan Diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 83.

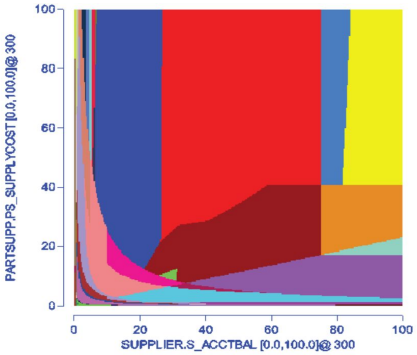


Figure 8: Plan diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 46.

EQCO: Introduction

PingCAP

Link: Exact Cardinality Query Optimization for Optimizer Testing

Motivation: exact cardinalities are very valuable in optimizer testing:

1. cardinality-optimal plans can serve as a benchmark baseline;
2. help to narrow down the cause of a plan quality problem (decoupling CE from the optimizer let us can test CE and cost model separately);

Challenge: the natural approach of executing all relevant sub-expressions to obtain cardinalities can be expensive.

Introduction: propose a new algorithm that can significantly reduce the time of obtaining exact cardinalities.

This functionality has been implemented in Microsoft SQL Server.

Example 1. Consider the following query on the TPC-H database.

```
SELECT … FROM Lineitem, Orders, Customer
WHERE l_orderkey = o_orderkey
AND   o_custkey = c_custkey
AND   l_shipdate > '2008-01-01'
AND   l_receiptdate < '2008-02-01'
AND   l_discount < 0.05
AND   o_orderpriority = 'HIGH'
AND   c_mktsegment = 'AUTOMOBILE'
```

(1) (*l_shipdate* > '2008-01-01')
(2) (*l_receiptdate* < '2008-02-01')
(3) (*l_shipdate* > '2008-01-01' AND *l_receiptdate* < '2008-02-01')
(4) (*l_shipdate* > '2008-01-01' AND *l_receiptdate* < '2008-02-01' AND *l_discount* < 0.05)
(5) (*o_orderpriority* = 'HIGH')
(6) (*c_mktsegment* = 'AUTOMOBILE')
(7) (*Customer* ⋈ *Orders*)
(8) (*Orders* ⋈ *Lineitem*)
(9) (*Lineitem* ⋈ *Orders* ⋈ *Customer*)

EQCO: The Key Idea

PingCAP

The key idea: when an expression(query) is executed, we can obtain cardinalities of other related sub-expressions as a byproduct;

⇒ we can only execute a subset of expressions(queries) to obtain all relevent cardinalities.

The query and its relevant sub-expressions are optimizer dependent.

Example 1. Consider the following query on the TPC-H database.

```
SELECT … FROM Lineitem, Orders, Customer
WHERE l_orderkey = o_orderkey
AND   o_custkey = c_custkey
AND   l_shipdate > '2008-01-01'
AND   l_receiptdate < '2008-02-01'
AND   l_discount < 0.05
AND   o_orderpriority = 'HIGH'
AND   c_mktsegment = 'AUTOMOBILE'
```

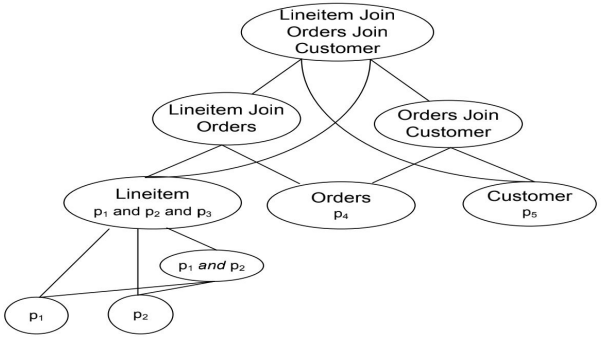


Figure 1. For a query on TPC-H database, expressions whose cardinalities are used by the query optimizer.

EQCO: For a Single Query

PingCAP

Convert this problem into a **Weighted Set Cover problem**.

SQL: select * from t1, t2 where t1.a>1 and t2.b<1 and t1.id=t2.id, all relevant expressions are:

1. t1.a>1 ⇒ select count(*) from t1 where t1.a>1;
2. t2.b<1 ⇒ select count(*) from t2 where t2.b<1;
3. t1.id=t2.id and t1.a>1 // IndexJoin(t1, t2)
4. t1.id=t2.id and t2.b<1 // IndexJoin(t2, t1)
5. t1.id=t2.id and t1.a>1 and t2.b<1 // predicates are pushed down

coverage relationship:

1. 1 ⇒ 1
2. 2 ⇒ 2
3. 3 ⇒ 1, 3
4. 4 ⇒ 2, 4
5. 5 ⇒ 1, 2, 5

a greedy algorithm based on their coverage relationship and costs

CoveringQueriesOptimization

Input: Set of relevant expressions R_Q for a query Q

Output: $\mathbf{R} \subseteq \mathbf{R_Q}$ such that executing all expressions in **R** gives exact cardinalities for all expression in $\mathbf{R_Q}$

1. $\mathbf{R} = \{\}, S = \{\}$
2. **While** $(S \mathrel{!}= \mathbf{R_Q})$ **Do**
3. Pick $e \in (\mathbf{R_Q} - \mathbf{R})$ with the largest value of $|Exprs(e) - S|/Cost(e)$
4. $R = R \cup \{e\}; S = S \cup Exprs(e)$
5. **End While**
6. Return **R**

Figure 5. Algorithm for Covering Queries Optimization.

EQCO: For a workload

PingCAP

Some queries in a worklaod have a same signature;

We can use CASE-WHEN SQLs to compute multiple expressions in **a single pass**.

```
e1 = SELECT … FROM Lineitem
     WHERE l_discount < 0.05 and
           l_shipdate < '1998-01-01'

e2 = SELECT … FROM Lineitem
     WHERE l_discount < 0.15 and
           l_shipdate < '1997-01-01'
```

```
SELECT SUM(a) as card1, SUM(b) as card2
FROM
(
  SELECT a = CASE when (l_discount < 0.05
         and l_shipdate< '1998-01-01') then
         1 else 0 end,
         b = CASE when (l_discount < 0.15
         and l_shipdate < '1997-01-01') then
         1 else 0 end
  FROM Lineitem
  WHERE l_discount < 0.15 and
        l_shipdate < '1998-01-01'
)
```

EQCO: Candidate Generation

PingCAP

The Candidate Generation Algorithm:

```
SELECT … FROM Lineitem, Orders
WHERE l_orderkey = o_orderkey AND
   l_discount < 0.05 AND
   o_orderdate < '1997-01-01'
SELECT … FROM Lineitem, Orders
WHERE l_orderkey = o_orderkey AND
   l_discount < 0.15 AND
   o_orderdate < '1998-01-01'
```

o_orderdate<'1997-01-01'
or
o_orderdate<'1998-01-01'

```
Q: SELECT SUM(a) as card1, SUM(b) as card2
FROM (
 SELECT
 a = CASE WHEN l_discount < 0.05 AND
     o_orderdate < '1997-01-01'
     then 1 else 0 end,
 b = CASE WHEN l_discount < 0.15 AND
     o_orderdate < '1998-01-01'
     then 1 else 0 end
 FROM Lineitem, Orders
 WHERE l_orderkey = o_orderkey
 AND l_discount < 0.15
 AND o_orderdate < '1998-01-01'
)
```

EQCO: Candidate Selection

PingCAP

Which candidates should be selected for a given signature:

e1: select * from t where a<1 and b<1000

e2: select * from t where a<1000 and b<1

e12: select * from t where a<1 and b<1

Which to choose: {e1, e2} or {e12}?

Convert this problem to a **Weighted Set Cover problem.**

Use the similar greedy algorithm to solve it.

Benefit(e) = |Coverage(e)| / Cost(e)

SelectCASEQueryForSignature

Input: Set of expressions **R** belonging to a particular signature

Output: Candidate expression for signature

```
1.  Let u = candidate obtained by invoking
    GenerateCandidateQuery(R) // (least upper bound of the
    lattice)
2.  Do
3.    BetterNodeExists = false
4.    For each child expression e of u in the lattice
5.      GenerateCandidateQuery(e)
6.      If Benefit(e) > BestBenefit
7.          BestBenefit = Benefit(e); u = e;
8.          BetterNodeExists =true;
9.  While(BetterNodeExists)
10. Return u
```

Figure 9. Algorithm for selecting the a candidate expression for a given signature

EQCO: Effectiveness

Figure 11: for all single queries in TPC-H

Figure 12: for the workload TPC-H

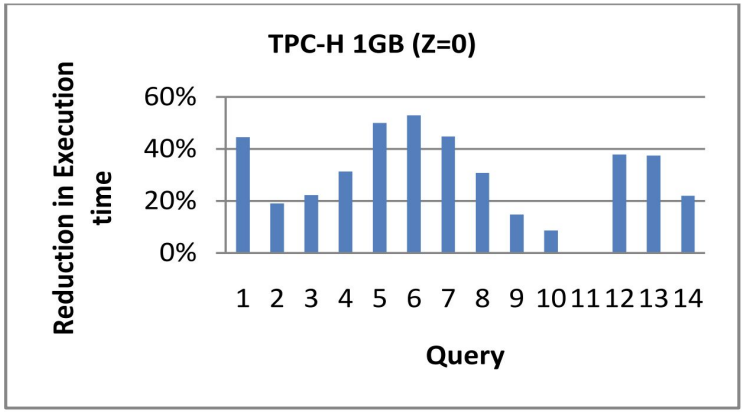


Figure 11. Impact on Covering Queries optimization for TPC-H queries.

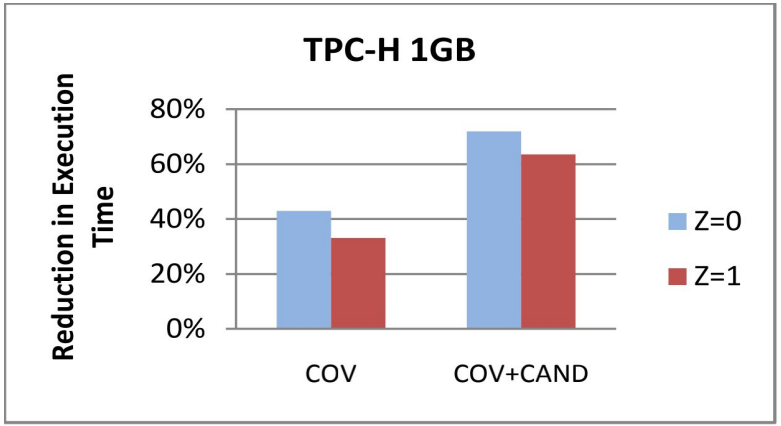


Figure 12. Performance of Covering Queries and Candidate Generation techniques for TPC-H workload.

EQCO: Analytics

Use exact cardinalities to benchmark CE Model and use Relative Error as the metric:

$$Relative\ Error(e) = \frac{|Actual(e) - Estimated(e)|}{Estimated(e)}$$

What we found:

1. larger errors as the number of tables in the expression increases;
2. some Group-By expresssions with single table also incur large error, because the statistics available in the DBMS are not adequate to capture this correlation;

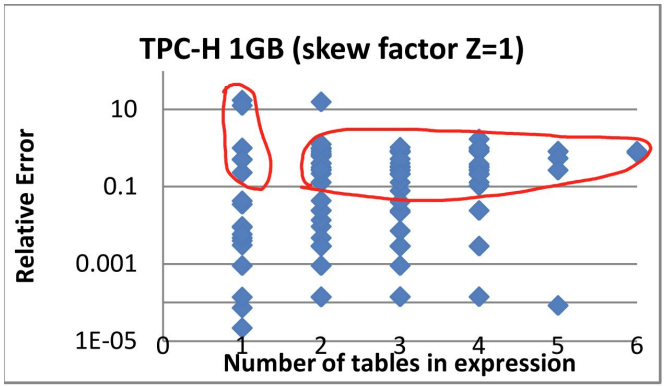


Figure 16. Cardinality estimation errors grouped by number of tables in the expression.

CE Testing: Introduction

PingCAP

Link: Testing Cardinality Estimation Models in SQL Sever
Introduction: Describe an exercise that the replacement and validation of a new CE Model in Miscrosoft SQL Sever.
Background:

- **Motivation**: after multiple releases, ad-hoc extensions, special case fixes, the CE component became very hard to debug, predict, or understand
- **The goals of this rewrite**:
    1. separate the tasks of deciding how to compute an estimate and actually performing the computation;
    2. allow the fundamental assumption to be easily changed(uniformity, independence, containment);
    3. incorporate prior extensions into the model as first class citizens to make it more predictable and reasonable;

Quality assurance process:

1. Customer Workloads and Benchmarks
2. Synthetic Functional Testing to ensure functional correctness:
    a. large-scale stochastic testing
    b. extensions to the existing estimation quality and stability tests
    c. targeted tests that validate the behavior of the new model with respect to the underlying assumptions it made

CE Testing: Model Variations

PingCAP

Model Variations: use a more formalized concept to deal with special cases in the old CE model.
Advantages: This design allowed us to conditionally enable and siable variations on demand, which is very usefule for testing;

Examples of model variations:

1. selectivity computation for conjunction:
    a. assume complete correlation between predicates
    b. choose the minimul of the individual selectivities
2. join size estimation: where t1.id=t2.id and cond(t1)
    a. overpopulated PK: The rationale is that when the primary key values are filtered prior to the join, the filter is typically biased towards removing those values that are not present on the foreign key side
    b. simple join: simply employ a containment model, `result=|t1|*|t2|/NDV(t1)`

CE Testing: Customer Workloads

Dataset: in-house cutomer queries + standard benchmarks + real customer workloads; (30+ different workloads)
The metric: query response time + query compile-time + compile-time memory footprint;
Through careful combining of model variation: the new CE model surpasses the old in most cases.

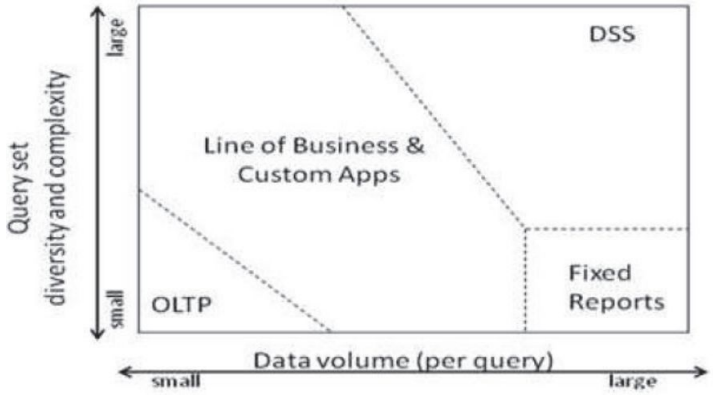


Figure 1: A categorization of database application space.

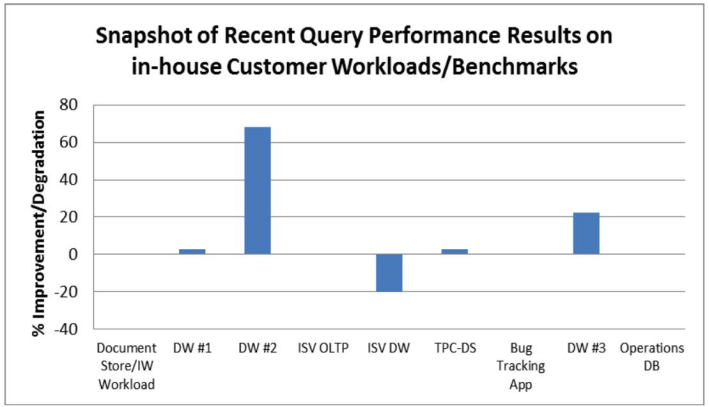


Figure 3: Snapshot of query performance (response times) comparison between old and new CE on a recent build.

CE Testing: Targeted Testing

Motivation: validate the behavior of the new model with respect to the underlying assumptions it made;
("over-fitted"? "two wrongs make right"?)

An example of two-table one-to-many join:

```
SELECT Count(*)
FROM   Dim, Fact
WHERE  Dim.dimId = Fact.refDimId and
       Dim.dimValue <= {constant}
```

Fact.refDimId is populated using values from Dim.dimId.
Adjust the degree of containment and observe whether models work as expectation.

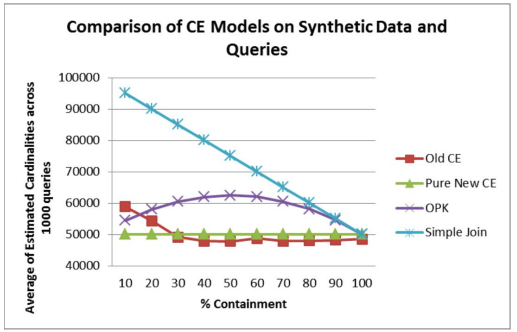


Figure 4: Characterizing estimations made by four estimation models.

Summary:

1. chose a "winning" CE model by evaluating perf of variations agains at real customer workloads and benchmarks;
2. useing targeted testing to validate the behavior of the winning model against its underlying assumptions;
3. to ensure the model implementation is reliable, performed large-scale stochastic testing;

CE Testing: Plan Diagram

This is a realistic practice in SQL Sever Team when refactoring the CE Model.

Snowy Plan Diagram(Figure 7) is caused by floating point inconsistent in the code

This reduction in distinct plans will manifest itself in less volatile query plan selection

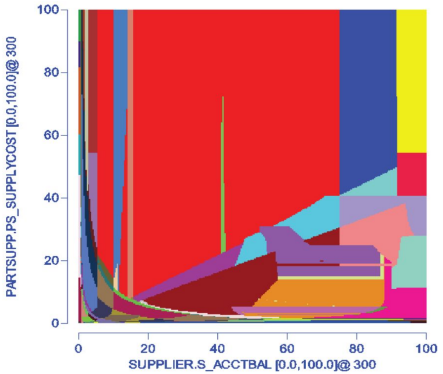


Figure 6: Plan diagram of TPC-H, Query 9, for the old cardinality estimation model. Number of distinct plans: 78.

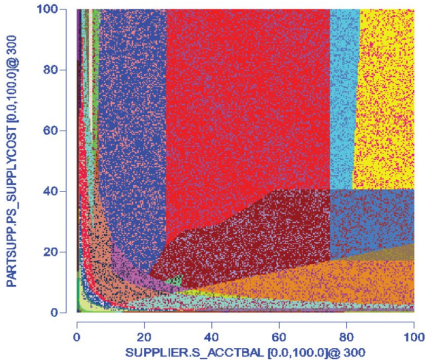


Figure 7: Original (Snowy) Plan Diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 83.

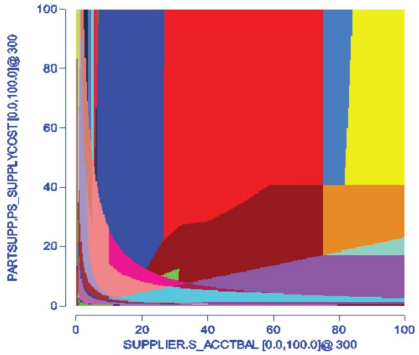


Figure 8: Plan diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 46.

TAQO: Introduction

PingCAP

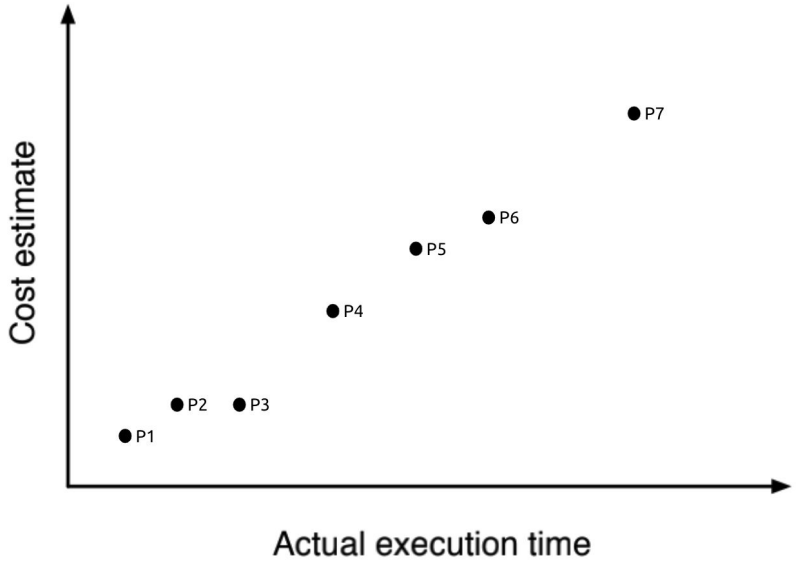
Link: Testing the Accuracy of Query Optimizers

Introduction: a framework(method) to quantify an optimizer's accuracy for a given workload.

The key idea:

The cost model itself is hardware/optimizer dependent(cost unit/formula), but the correlation between costs and execution times should be highly consistent.

Define accuracy as the cost model's ability to order any two given plans correctly(the plan with the higher cost will run longer).



TAQO: Rank Correlation Metric

PingCAP

Convert the key idea to a specific formula, and some factors should be considered:

1. **discordance of plan pairs**: the metric penalizes discordant paris of plans by using Kendall's Tau
2. **relevance of plan**: the metric penalizes ranking errors involving important plans (close to the optimal plan)
3. **pairwise distance**: if the actual execution cost for two plans is very close, we might not be able to order them

p[i] = (a[i], e[i]), S = [p[1], p[2]... p[n]] order by a[i];

$$\tau = \sum_{i<j} \mathrm{sgn}(e_j - e_i)$$

(-1, 0, 1) or (0, 1)?

$$w_m = \frac{a_1}{a_m}$$

$$d_{ij} = \sqrt{\left(\frac{a_j - a_i}{a_n - a_1}\right)^2 + \left(\frac{e_j - e_i}{\max_k(e_k) - \min_k(e_k)}\right)^2}$$

$$s = \sum_{i<j} w_i w_j d_{ij} \cdot \mathrm{sgn}(e_j - e_i)$$

relevance distance discordance

It's a penalty, so the lower the value of the metric, the higher the accuracy of the optimier.

TAQO: Optimizers Comparison

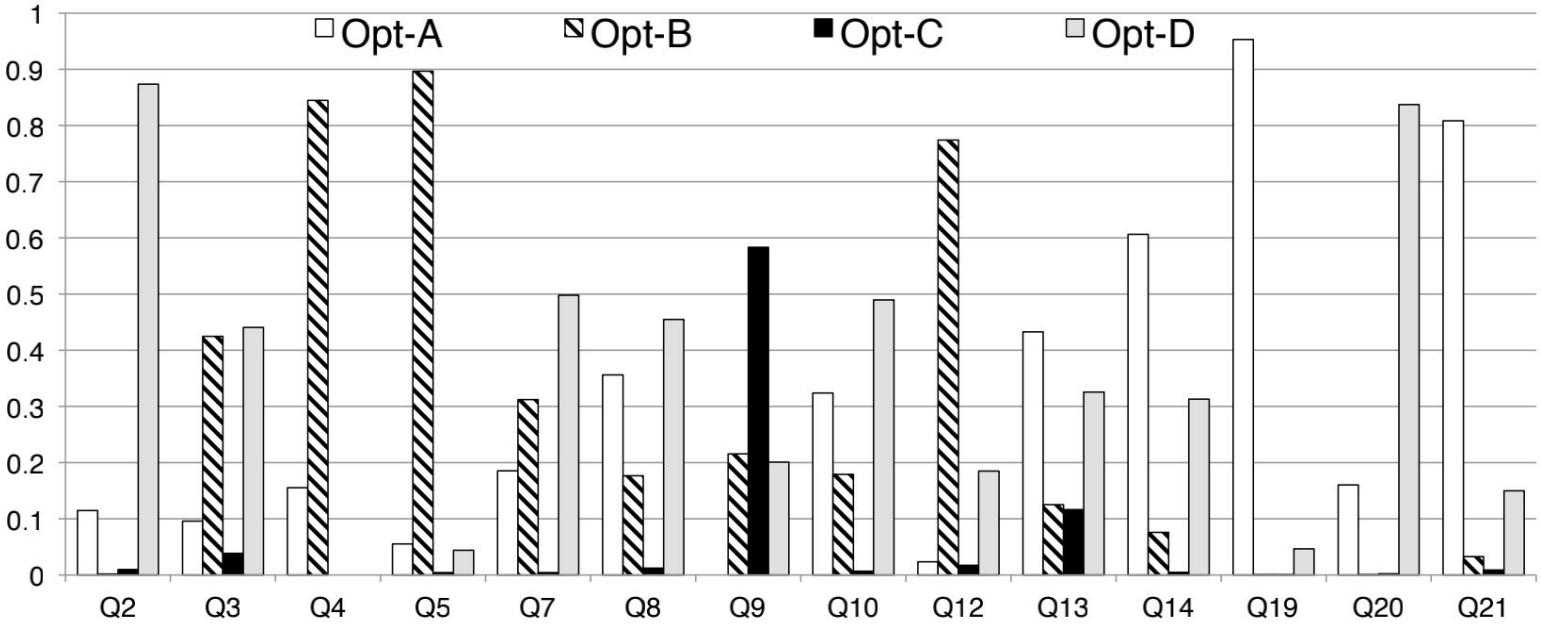


Figure 3: Correlation scores for different TPC-H queries. Low score indicates high accuracy

TAQO: Analysis and Summary

When an optimizer consistently ranks plan alternatives correctly, the plan distribution plot shows a monotonic trend. (opt-C)

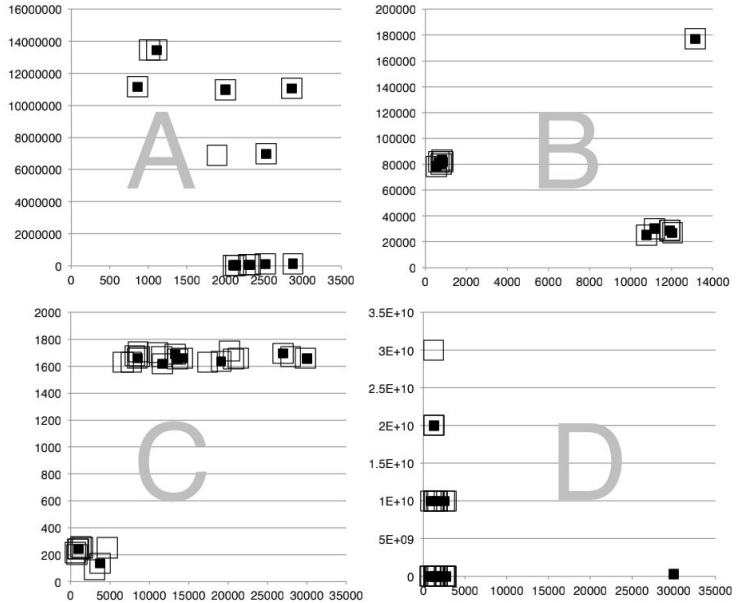


Figure 6: Plan distribution plots for $Q14$. Actual execution time is on the x-axis, and estimated cost is on the y-axis

GQO: Introduction

Link: How Good Are Optimizers, Really?

Motivation: Three questions:

1. How good are cardinality estimators and when do bad estimates lead to slow queries?
2. How import is an aacurate cost model for the overall query optimization process?
3. How large does the enumerated plan space need to be?

Introduction: This is an experiments and analysese paper:

1. a novel methodology to isolate the influence of the individual component
2. some useful guidelines for optimizer designers

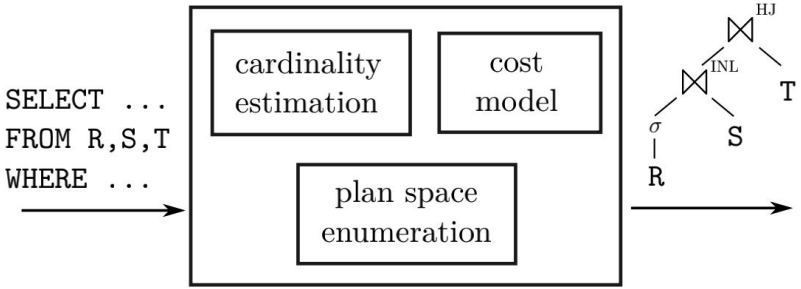


Figure 1: Traditional query optimizer architecture

GQO: IMDB + JOB

Why not use standard benchmarks (TPC-H / TPC-DS / SSB)?

> *In order to **easily be able to scale** the benchmark data, the data generators are using the very same **simplifying assumptions** (uniformity, independence, principle of inclusion), while real-world data sets are full of **correlations and non-uniform data distributions**.*

The Internet Movie Data Base(IMDB): **realistic, full of correlations**

Based on IMDB, analytical queries are constructed:

1. **complex**: 3-16 joins, 113 queries
2. **realistic**: that may be asked by a movie enthusiast:
   a. q-13d: “the ratings and release dates for all movies produced by US companies”

```
SELECT cn.name, mi.info, miidx.info
FROM company_name cn, company_type ct,
     info_type it, info_type it2, title t,
     kind_type kt, movie_companies mc,
     movie_info mi, movie_info_idx miidx
WHERE cn.country_code ='[us]'
AND ct.kind = 'production companies'
AND it.info = 'rating'
AND it2.info = 'release dates'
AND kt.kind = 'movie'
AND ... -- (11 join predicates)
```

GQO: Estimates for JOB & TPCH

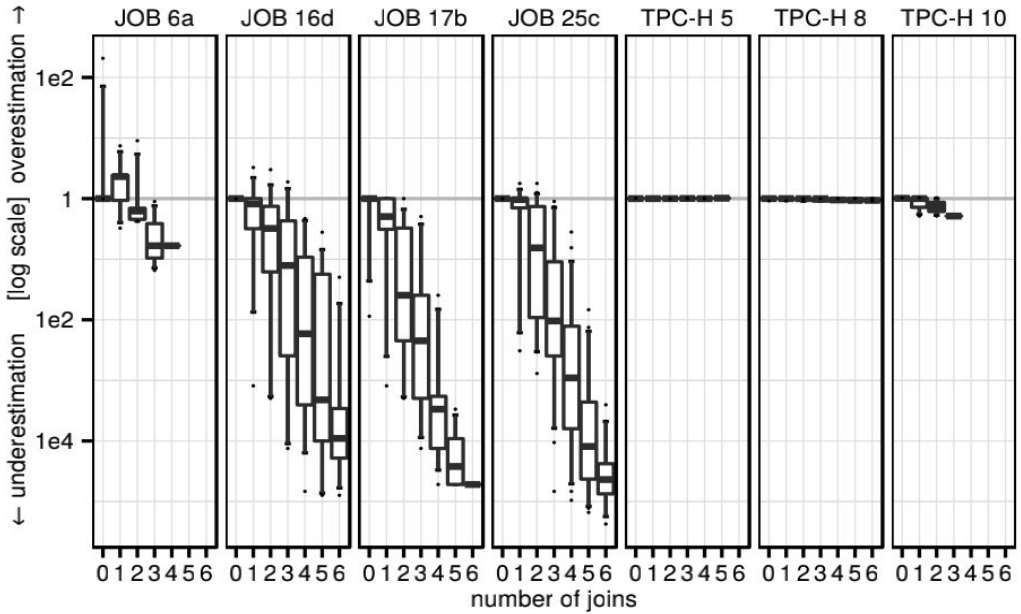


Figure 4: PostgreSQL cardinality estimates for 4 JOB queries and 3 TPC-H queries

GQO: Cardinality Extraction

Extract cardinality estimates for all intermediate results from 5 relational DBs:

1. PG
2. Hyper
3. 3 commercial systems

An example: select * from A, B, C where A.bid=B.id and B.cid=c.id and A.x=5

- where A.x=5
- where A.x=5 and A.bid=B.id
- where B.cid=c.id
- where A.bid=B.id and B.cid=c.id and A.x=5

Cardinality Injection:

1. Inject cardinalities from different systems into PG to **avoid the influence of execution engine**.
2. Injecting and using true cardinalities can **avoid the Influence of wrong estimation**.

GQO: The impact of cardinality misestimation

The experiment:

- inject the estimates of different systems into PG,
- then execute the resulting plans on PG,
- and then compare their runtime with true cardinalities;

	<0.9	[0.9,1.1)	[1.1,2)	[2,10)	[10,100)	>100
PostgreSQL	1.8%	38%	25%	25%	5.3%	5.3%
DBMS A	2.7%	54%	21%	14%	0.9%	7.1%
DBMS B	0.9%	35%	18%	15%	7.1%	25%
DBMS C	1.8%	38%	35%	13%	7.1%	5.3%
HyPer	2.7%	37%	27%	19%	8.0%	6.2%

The influence of wrong cardinalities is huge:

- most queries in all systems are significantly slower (> 1.1)
- occasionally lead to plans that take an unreasonable time (> 10)
- NOTE: cases in (< 0.9) are caused by wrong cost model

GQO: Cost Models

Compare three cost models:

- the PG's cost model:
  - disk-oriented
  - combines CPU and I/O cost with certain weights
- tunning for main memory:
  - PG suggest that processing a tuple is 400x cheaper than reading it
  - multiplying the CPU cost parameters by a factor of 50 (400 / 50 = 8)
- a very simple cost model *Cmm*:
  - does not model I/O costs
  - only counts the number of tuples

$$C_{\text{mm}}(T) = \begin{cases} \tau \cdot |R| & \text{if } T = R \vee T = \sigma(R) \\ |T| + C_{\text{mm}}(T_1) + C_{\text{mm}}(T_2) & \text{if } T = T_1 \bowtie^{\text{HJ}} T_2 \\ C_{\text{mm}}(T_1) + & \text{if } T = T_1 \bowtie^{\text{INL}} T_2, \\ \quad \lambda \cdot |T_1| \cdot \max(\frac{|T_1 \bowtie R|}{|T_1|}, 1) & (T_2 = R \vee T_2 = \sigma(R)) \end{cases}$$

GQO: The Cost Model Experiment

The experiment:

- correlation between the cost and runtime
- the linear regression (good enough)

Key points:

- **dwarfed by compared with replacing the true cardinality**
- the median e: 38% in b ⇒ 30% in d;
- geometric mean runtime over all queries in d is 41% faster than b, and f is 34% faster than b;

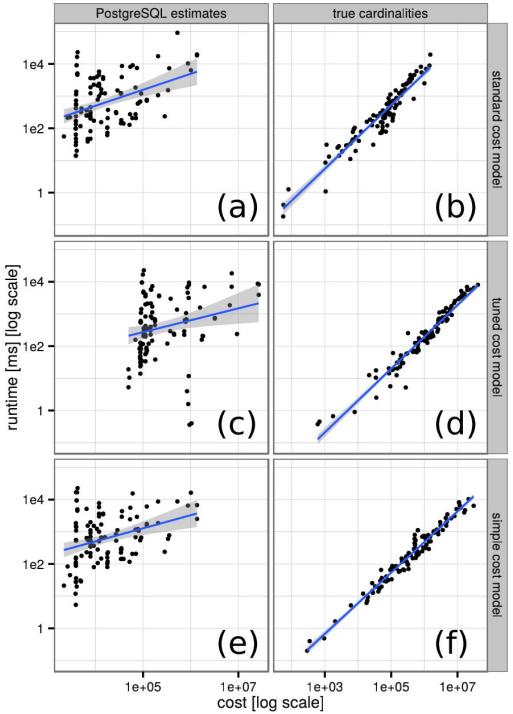


Figure 8: Predicted cost vs. runtime for different cost models

GQO: How Important Is the Join Order

The experiment:

- the Quickpick algorithm
- calculate costs with true cardinalities
- 10000 times per query

Key points:

- the slowest or median cost is multiple orders of magnitude more expensive than the cheapest;
- the optimal plan with more indexes is much faster than others;

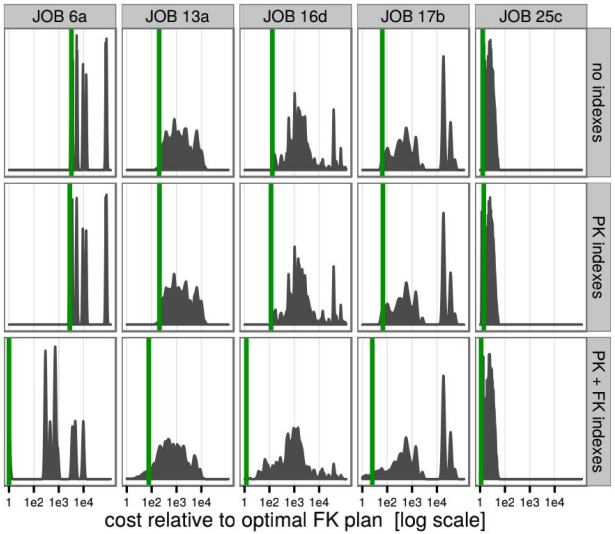


Figure 9: Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan

GQO: Are Heuristic Good Enough?

Compare three algorithm:

- the dynamic programming algorithm (DP)
- Quickpick-1000
- Greedy Operator Ordering (GOO)

	PK indexes						PK + FK indexes					
	PostgreSQL estimates			true cardinalities			PostgreSQL estimates			true cardinalities		
	median	95%	max	median	95%	max	median	95%	max	median	95%	max
Dynamic Programming	1.03	1.85	4.79	1.00	1.00	1.00	1.66	169	186367	1.00	1.00	1.00
Quickpick-1000	1.05	2.19	7.29	1.00	1.07	1.14	2.52	365	186367	1.02	4.72	32.3
Greedy Operator Ordering	1.19	2.29	2.36	1.19	1.64	1.97	2.35	169	186367	1.20	5.77	21.0

Table 3: Comparison of exhaustive dynamic programming with the Quickpick-1000 (best of 1000 random plans) and the Greedy Operator Ordering heuristics. All costs are normalized by the optimal plan of that index configuration

The performance potential from good cardinality estimates is certainly much larger!!

Quantitative impact: cardinality estimasion > plan space > cost model

GQO: Estimates for Base Tables

629 base selections:

- the median q-error is close to the optimal value 1 for all systems
- all systems produce misestimates

	median	90th	95th	max
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

Hyper can usually predict **complex predicates** well? (LIKE...)

⇒ use a random sample of 1000 rows per table

⇒ advantage: detect (anti-) correlations between attributes

⇒ bad cases: extremely low selectivities if no row on the sample

e.g: two attributes A and B, A is always equal to B*2; Sel(A=1 and B=2)?

- per-attribute histograms: Sel(A=1) * Sel(B=2)
- random-sampling: O_O

Summary

Analyzing Plan Diagrams of Database Query Optimizers

1. Plan Diagram
2. Plan Space & Plan Choice

Exact Cardinality Query Optimization for Optimizer Testing

1. ECQO -> Weighed Set Coverage Problem
2. Exact-cards are valuable in testing: exact-card-plan as a baseline, benchmark CE

Testing Cardinality Estimation Models in SQL Server

1. Model Variation
2. Customer Workloads(Performance) + Targeted Testing(Correctness)

Testing The Accuracy of Query Optimizers

1. Accuracy = The ability to order any two plans correctly (higher cost, run longer)

How Good Are Query Optimizers, Really?

1. IMDB + JOB
2. isolate components' influence and test them separately
3. CE > Plan Space > Cost Model

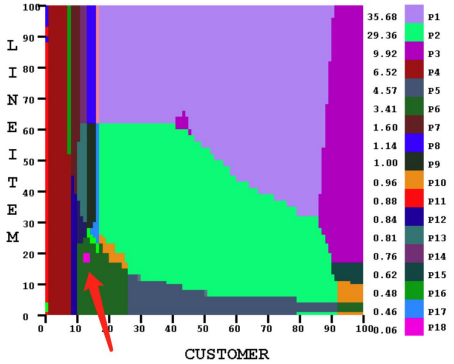
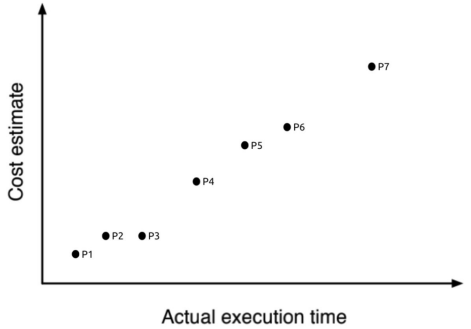


Figure 5: **Duplicates and Islands (Query 10, OptA)**



PingCAP

Thank You !
&
Discussion