Lecture #22

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Query Optimizer Cost Models

@Andy_Pavlo // 15-721 // Spring 2020

TODAY'S AGENDA

Cost Models

Cost Estimation

COST-BASED QUERY PLANNING

Generate an estimate of the cost of executing a particular query plan for the current state of the database.

→ Estimates are only meaningful internally.

This is independent of the search strategies that we talked about last class.

COST MODEL COMPONENTS

Choice #1: Physical Costs

→ Predict CPU cycles, I/O, cache misses, RAM consumption, pre-fetching, etc…

→ Depends heavily on hardware.

Choice #2: Logical Costs

→ Estimate result sizes per operator.

→ Independent of the operator algorithm.

→ Need estimations for operator result sizes.

Choice #3: Algorithmic Costs

→ Complexity of the operator algorithm implementation.

DISK-BASED DBMS COST MODEL

The number of disk accesses will always dominate the execution time of a query.

→ CPU costs are negligible.
→ Have to consider sequential vs. random I/O.

This is easier to model if the DBMS has full control over buffer management.

→ We will know the replacement strategy, pinning, and assume exclusive access to disk.

POSTGRES COST MODEL

Uses a combination of CPU and I/O costs that are weighted by "magic" constant factors.

Default settings are obviously for a disk-resident database without a lot of memory:

→ Processing a tuple in memory is **400x** faster than reading a tuple from disk.
→ Sequential I/O is **4x** faster than random I/O.

19.7.2. Planner Cost Constants

The *cost* variables described in this section are measured on an arbitrary scale. Only their relative values matter, hence scaling them all up or down by the same factor will result in no change in the planner's choices. By default, these cost variables are based on the cost of sequential page fetches; that is, `seq_page_cost` is conventionally set to `1.0` and the other cost variables are set with reference to that. But you can use a different scale if you prefer, such as actual execution times in milliseconds on a particular machine.

> **Note:** Unfortunately, there is no well-defined method for determining ideal values for the cost variables. They are best treated as averages over the entire mix of queries that a particular installation will receive. This means that changing them on the basis of just a few experiments is very risky.

`seq_page_cost` (`floating point`)

Sets the planner's estimate of the cost of a disk page fetch that is part of a series of sequential fetches. The default is 1.0. This value can be overridden for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name (see ALTER TABLESPACE).

`random_page_cost` (`floating point`)

IBM DB2 COST MODEL

Database characteristics in system catalogs

Hardware environment (microbenchmarks)

Storage device characteristics (microbenchmarks)

Communications bandwidth (distributed only)

Memory resources (buffer pools, sort heaps)

Concurrency Environment

→ Average number of users

→ Isolation level / blocking

→ Number of available locks

Source: [Guy Lohman]

IN-MEMORY DBMS COST MODEL

No I/O costs, but now we have to account for CPU and memory access costs.

Memory cost is more difficult because the DBMS has no control cache management.

→ Unknown replacement strategy, no pinning, shared caches, non-uniform memory access.

The number of tuples processed per operator is a reasonable estimate for the CPU cost.

SMALLBASE COST MODEL

Two-phase model that automatically generates hardware costs from a logical model.

Phase #1: Identify Execution Primitives

→ List of ops that the DBMS does when executing a query

→ Example: evaluating predicate, index probe, sorting.

Phase #2: Microbenchmark

→ On start-up, profile ops to compute CPU/memory costs

→ These measurements are used in formulas that compute operator cost based on table size.

MODELLING COSTS FOR A MM-DBMS
REAL-TIME DATABASES 1996

SELECTIVITY

The **selectivity** of an operator is the percentage of data accessed for a predicate.

→ Modeled as probability of whether a predicate on any given tuple will be satisfied.

The DBMS estimates selectivities using:

→ Domain Constraints
→ Precomputed Statistics (Zone Maps)
→ Histograms / Approximations
→ Sampling

OBSERVATION

The number of tuples processed per operator depends on three factors:

→ The access methods available per table
→ The distribution of values in the database's attributes
→ The predicates used in the query

Simple queries are easy to estimate.
More complex queries are not.

APPROXIMATIONS

Maintaining exact statistics about the database is expensive and slow.

Use approximate data structures called **sketches** to generate error-bounded estimates.

→ Count Distinct
→ Quantiles
→ Frequent Items
→ Tuple Sketch

See Yahoo! Sketching Library

SAMPLING

Execute a predicate on a random sample of the target data set.

The # of tuples to examine depends on the size of the table.

Approach #1: Maintain Read-Only Copy

→ Periodically refresh to maintain accuracy.

Approach #2: Sample Real Tables

→ Use **READ UNCOMMITTED** isolation.

→ May read multiple versions of same logical tuple.

RESULT CARDINALITY

The number of tuples that will be generated per operator is computed from its selectivity multiplied by the number of tuples in its input.

RESULT CARDINALITY

Assumption #1: Uniform Data
- → The distribution of values (except for the heavy hitters) is the same.

Assumption #2: Independent Predicates
- → The predicates on attributes are independent

Assumption #3: Inclusion Principle
- → The domain of join keys overlap such that each key in the inner relation will also exist in the outer table.

CORRELATED ATTRIBUTES

Consider a database of automobiles:

→ # of Makes = 10, # of Models = 100

And the following query:

→ `(make="Honda" AND model="Accord")`

With the independence and uniformity assumptions, the selectivity is:

→ $1/10 \times 1/100 = 0.001$

But since only Honda makes Accords the real selectivity is $1/100 = 0.01$

Source: Guy Lohman

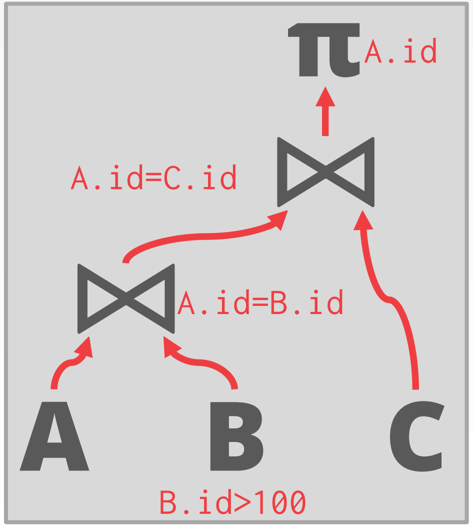
COLUMN GROUP STATISTICS

The DBMS can track statistics for groups of attributes together rather than just treating them all as independent variables.

→ Only supported in commercial systems.
→ Requires the DBA to declare manually.

ESTIMATION PROBLEM

```sql
SELECT A.id
  FROM A, B, C
 WHERE A.id = B.id
   AND A.id = C.id
   AND B.id > 100
```



Compute the cardinality of base tables

$A \rightarrow |A|$

$B.id>100 \rightarrow |B| \times sel(B.id>100)$

$C \rightarrow |C|$

Compute the cardinality of join results

$$A \bowtie B = (|A| \times |B|) \ / \ max(sel(A.id=B.id), sel(B.id>100))$$

$$(A \bowtie B) \bowtie C = (|A| \times |B| \times |C|) \ / \ max(sel(A.id=B.id), sel(B.id>100), sel(A.id=C.id))$$

ESTIMATOR QUALITY

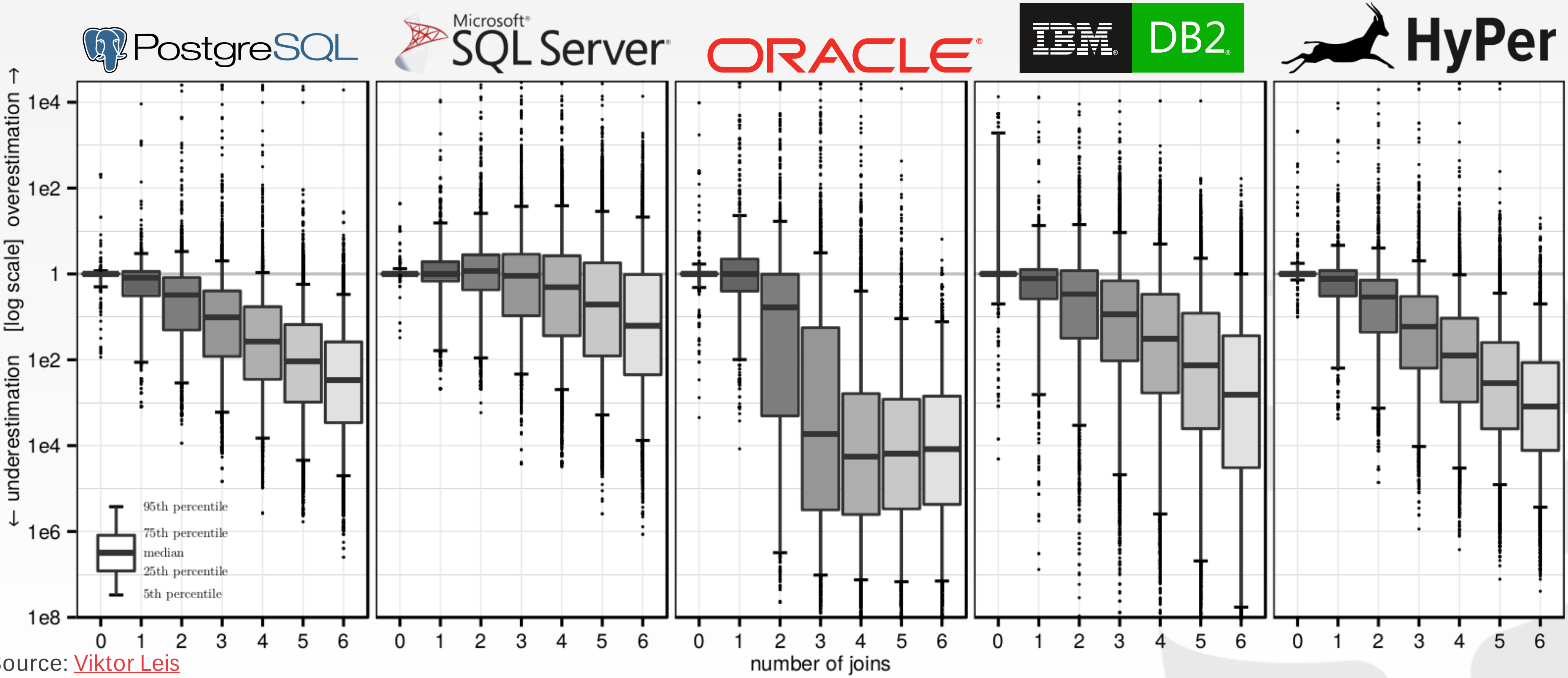
Evaluate the correctness of cardinality estimates generated by DBMS optimizers as the number of joins increases.

→ Let each DBMS perform its stats collection.
→ Extract measurements from query plan.

Compared five DBMSs using 100k queries.

HOW GOOD ARE QUERY OPTIMIZERS, REALLY?
VLDB 2015

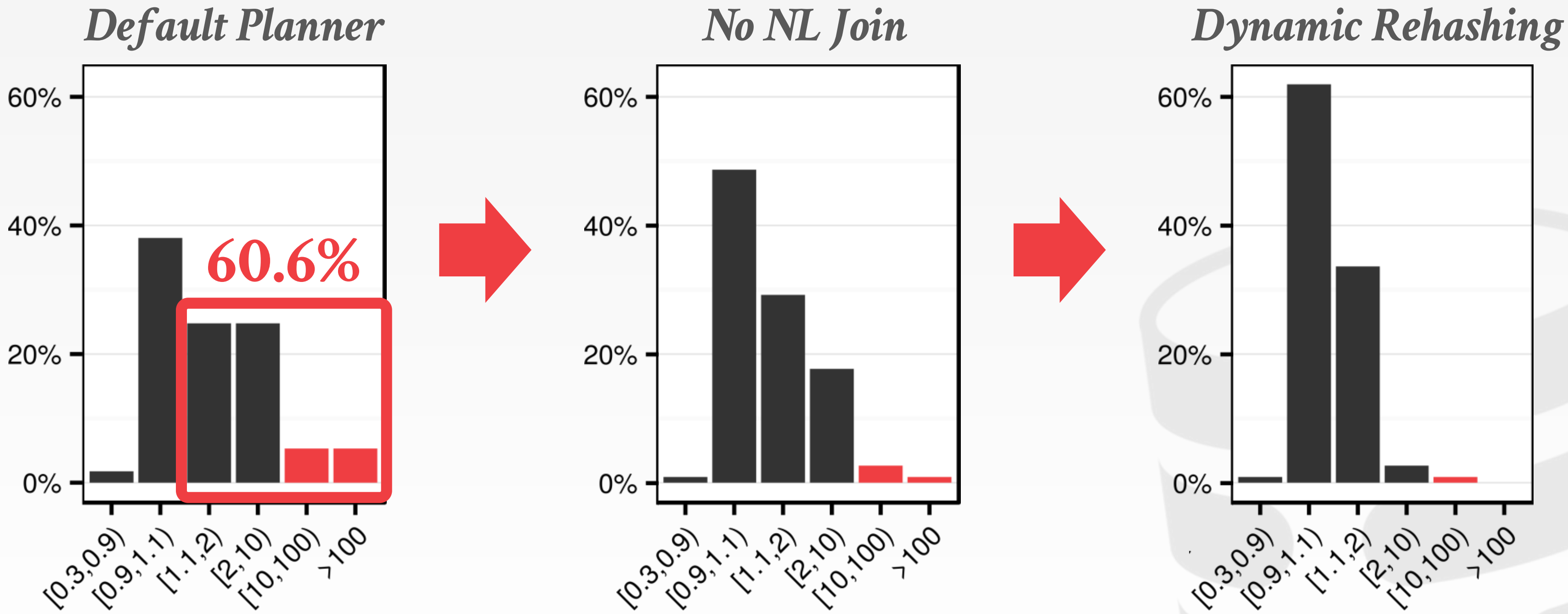
ESTIMATOR QUALITY



Source: Viktor Leis

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload



Source: Viktor Leis

Slowdown compared to using true cardinalities

LESSONS FROM THE GERMANS

Query opt is more important than a fast engine
- → Cost-based join ordering is necessary

Cardinality estimates are routinely wrong
- → Try to use operators that do not rely on estimates

Hash joins + seq scans are a robust exec model
- → The more indexes that are available, the more brittle the plans become (but also faster on average)

Working on accurate models is a waste of time
- → Better to improve cardinality estimation instead

Source: Viktor Leis

PARTING THOUGHTS

Using number of tuples processed is a reasonable cost model for in-memory DBMSs.

→ But computing this is non-trivial.

I think that a combination of sampling + sketches are the way to achieve accurate estimations.