Georgia Tech

DATABASE SYSTEM IMPLEMENTATION

GT 4420/6422 // SPRING 2019 // @JOY_ARULRAJ

LECTURE #16: COST MODELS

CREATING THE NEXT®

LOGISTICS

Reminder: Assignment #3 due on Tue (Mar 12).

Reminder: Assignment #4 released today. Due on Tue (Apr 2).

TODAY'S AGENDA

Cost Models

Cost Estimation

Technical Writing

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`)

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

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.

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

IBM DB2 – LEARNING OPTIMIZER

Update table statistics as the DBMS scans a table during normal query processing.

Check whether the optimizer's estimates match what it encounters in the real data and incrementally updates them.

LEO - DB2'S LEARNING OPTIMIZER
VLDB 2001

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 × 1/100 = 0.001

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

Source: Guy Lohman

COLUMN GROUP STATISTICS

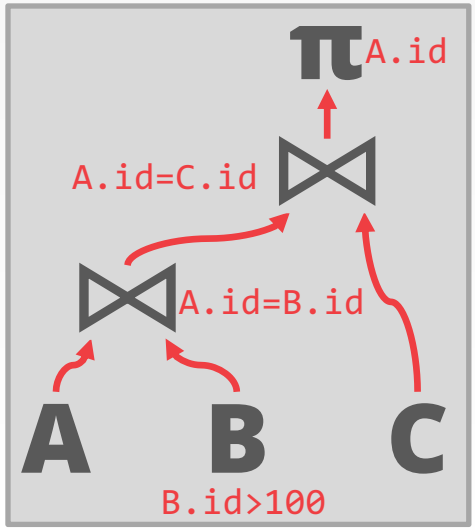
The DBMS can track statistics for groups of attributes together rather than just treating them all as independent variables.

→ Only supported in commercial systems.

→ Requires the DBA to declare manually.

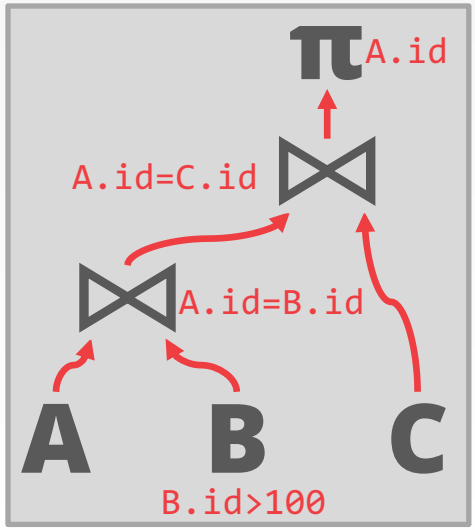
ESTIMATION PROBLEM

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



ESTIMATION PROBLEM

```
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

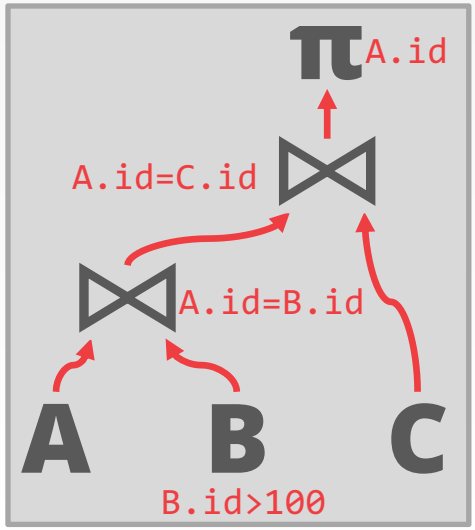
$$\mathbf{A} \rightarrow |\mathbf{A}|$$

$$\mathbf{B}.\mathrm{id}>100 \rightarrow |\mathbf{B}| \times \mathit{sel}(\mathbf{B}.\mathrm{id}>100)$$

$$\mathbf{C} \rightarrow |\mathbf{C}|$$

ESTIMATION PROBLEM

```
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

$$\mathbf{A} \rightarrow |\mathbf{A}|$$

$$\mathbf{B}.\mathrm{id}>100 \rightarrow |\mathbf{B}| \times \mathit{sel}(\mathbf{B}.\mathrm{id}>100)$$

$$\mathbf{C} \rightarrow |\mathbf{C}|$$

Compute the cardinality of join results

$$\mathbf{A} \bowtie \mathbf{B} = (|\mathbf{A}| \times |\mathbf{B}|) / \mathit{max}(\mathit{sel}(\mathbf{A}.\mathrm{id}=\mathbf{B}.\mathrm{id}), \mathit{sel}(\mathbf{B}.\mathrm{id}>100))$$

$$(\mathbf{A} \bowtie \mathbf{B}) \bowtie \mathbf{C} = (|\mathbf{A}| \times |\mathbf{B}| \times |\mathbf{C}|) / \mathit{max}(\mathit{sel}(\mathbf{A}.\mathrm{id}=\mathbf{B}.\mathrm{id}), \mathit{sel}(\mathbf{B}.\mathrm{id}>100), \mathit{sel}(\mathbf{A}.\mathrm{id}=\mathbf{C}.\mathrm{id}))$$

ESTIMATOR QUALITY

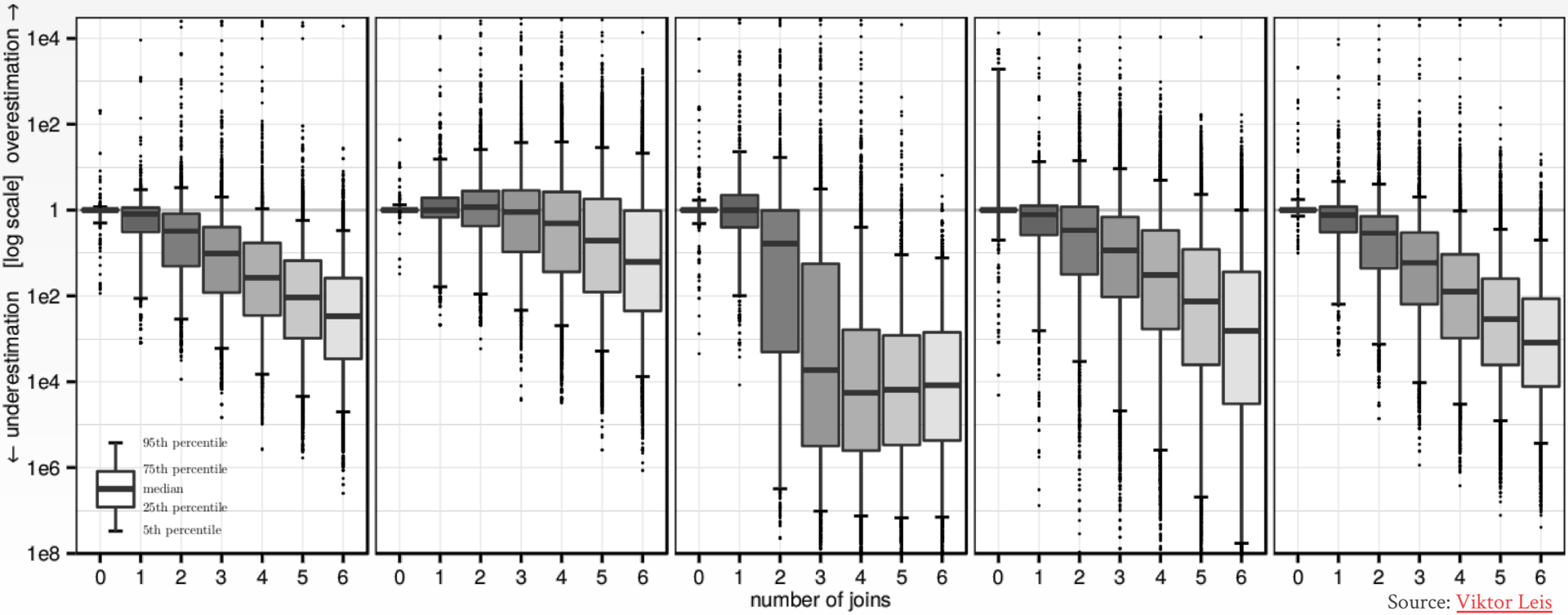
Evaluate the correctness of cardinality estimates generated by DBMS optimizers as the number of joins increases.

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

Compared five DBMSs using 100k queries.

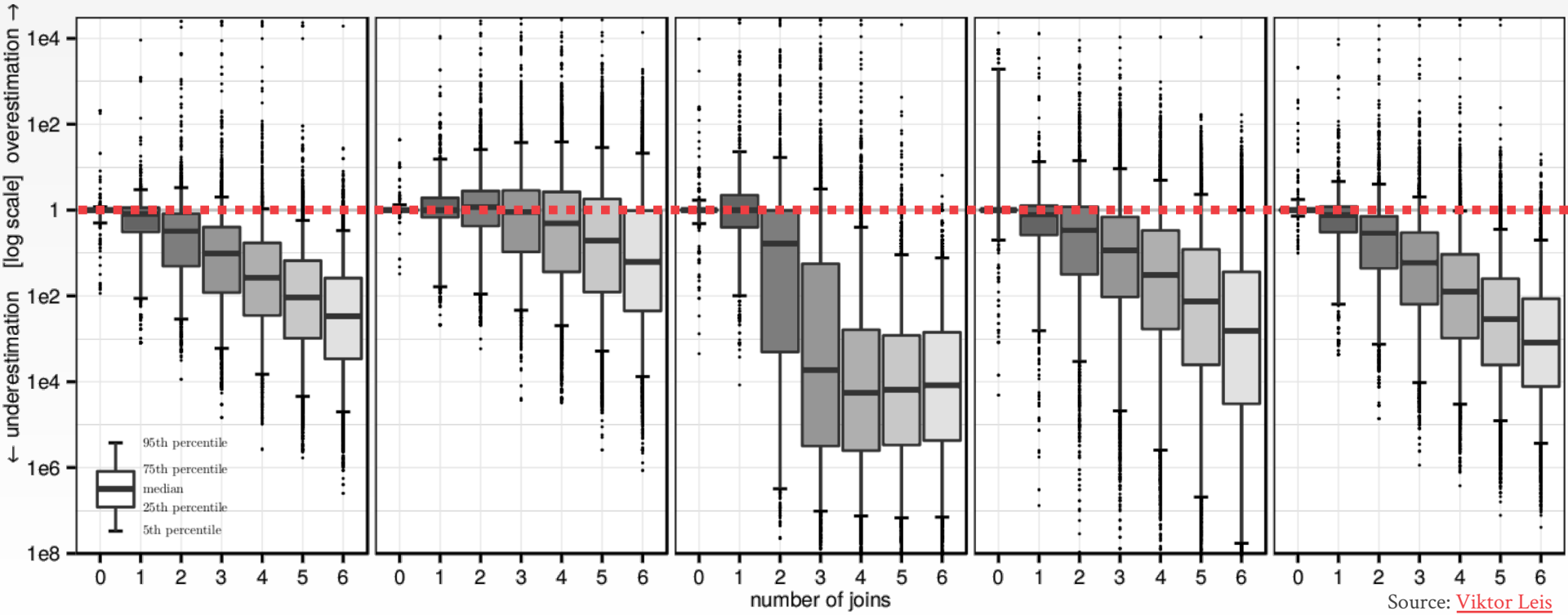
HOW GOOD ARE QUERY OPTIMIZERS, REALLY?
VLDB 2015

ESTIMATOR QUALITY



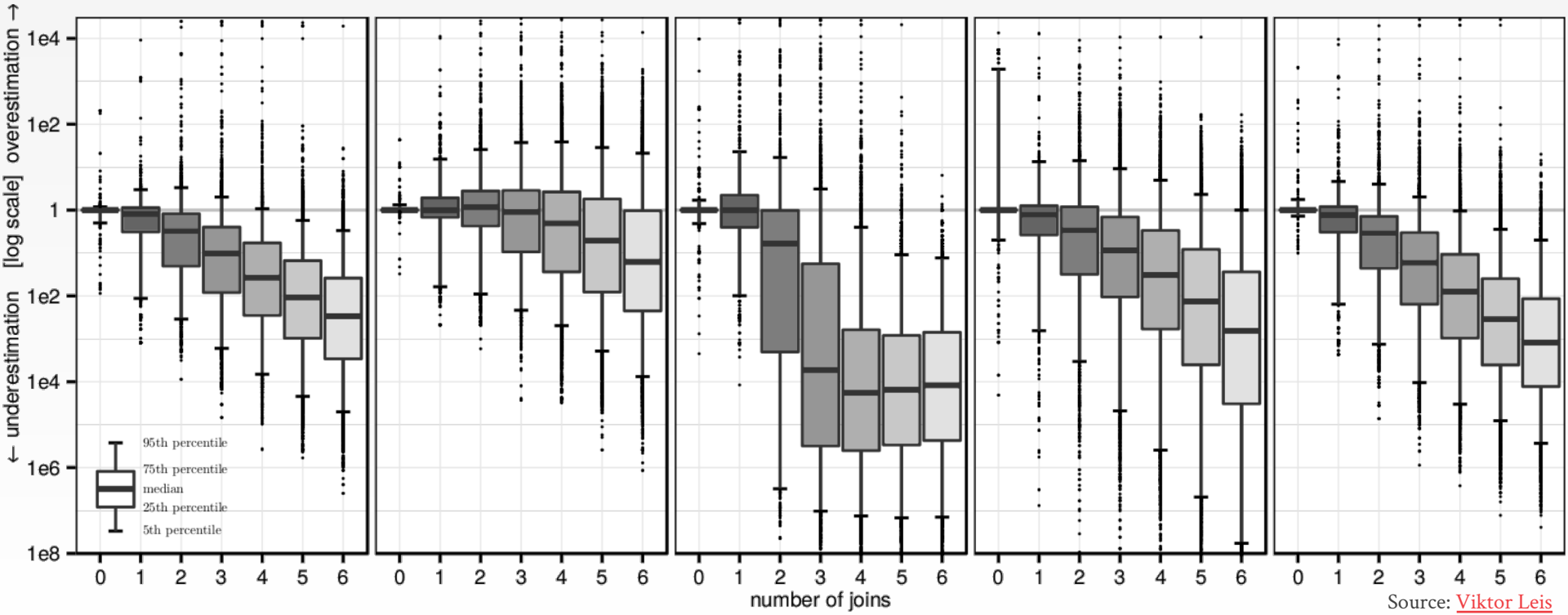
Source: Viktor Leis

ESTIMATOR QUALITY



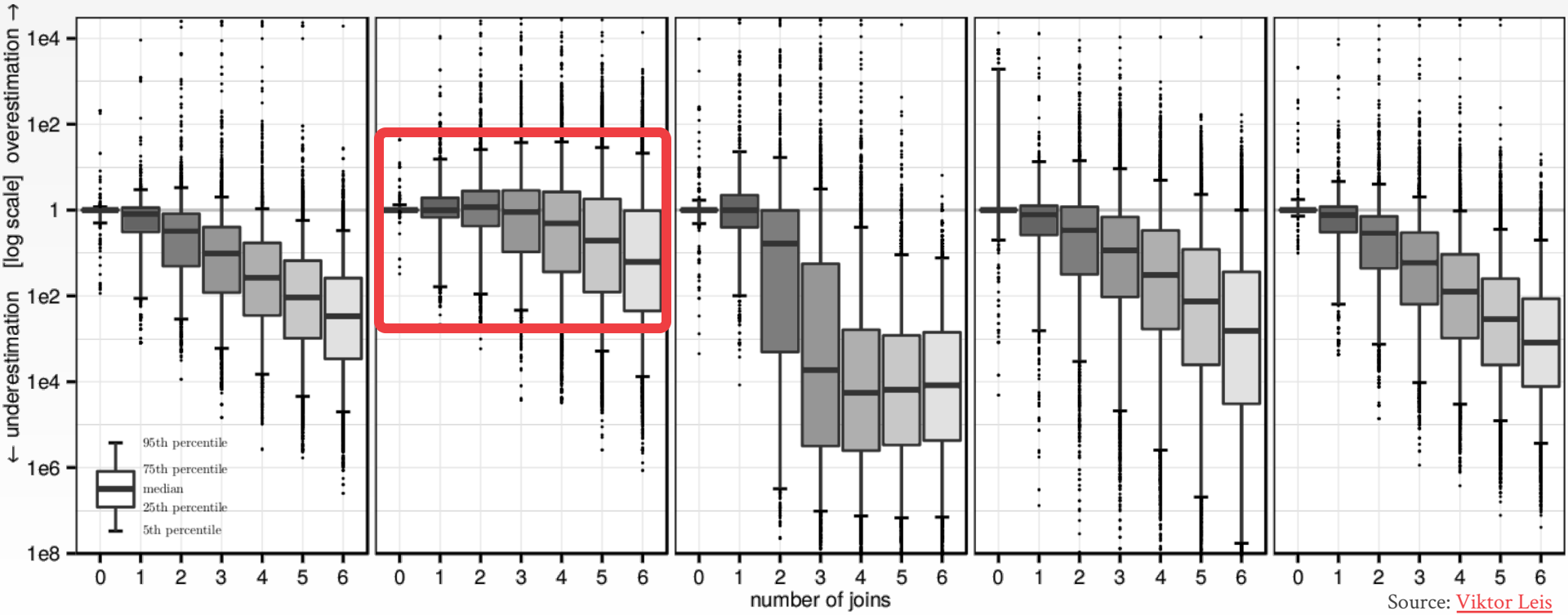
Source: Viktor Leis

ESTIMATOR QUALITY



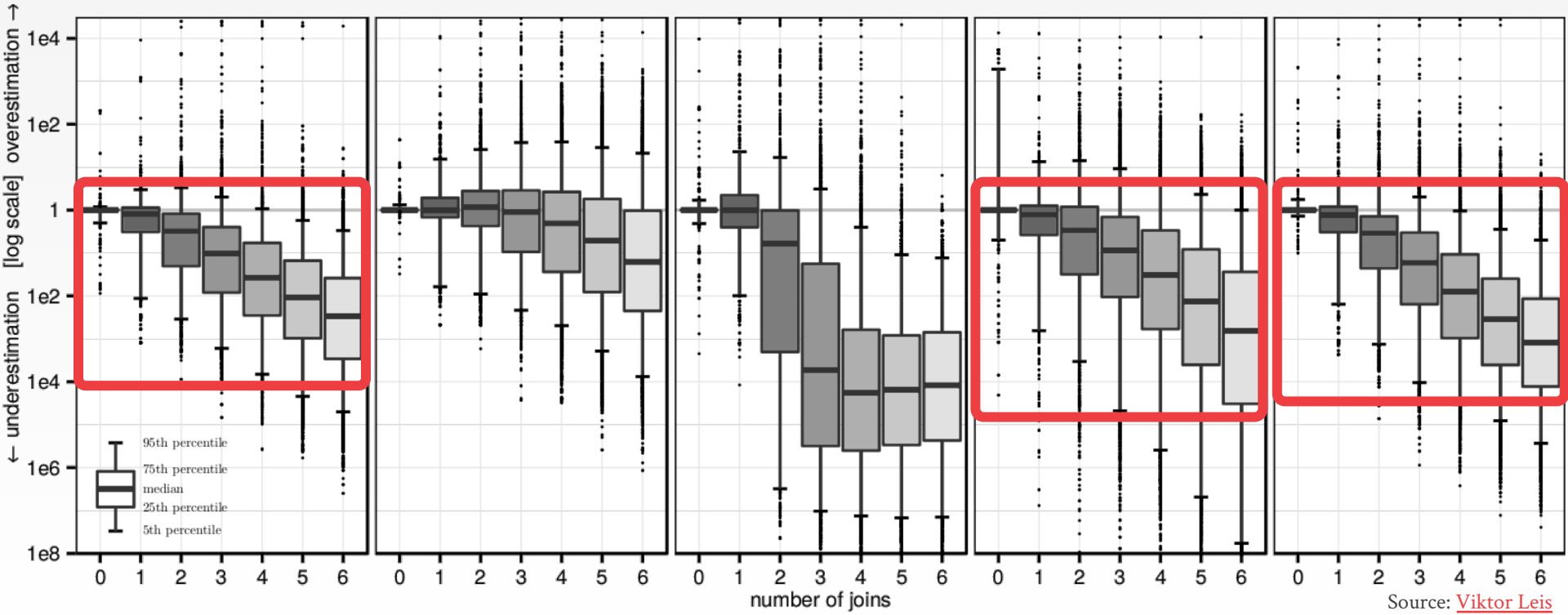
Source: Viktor Leis

ESTIMATOR QUALITY



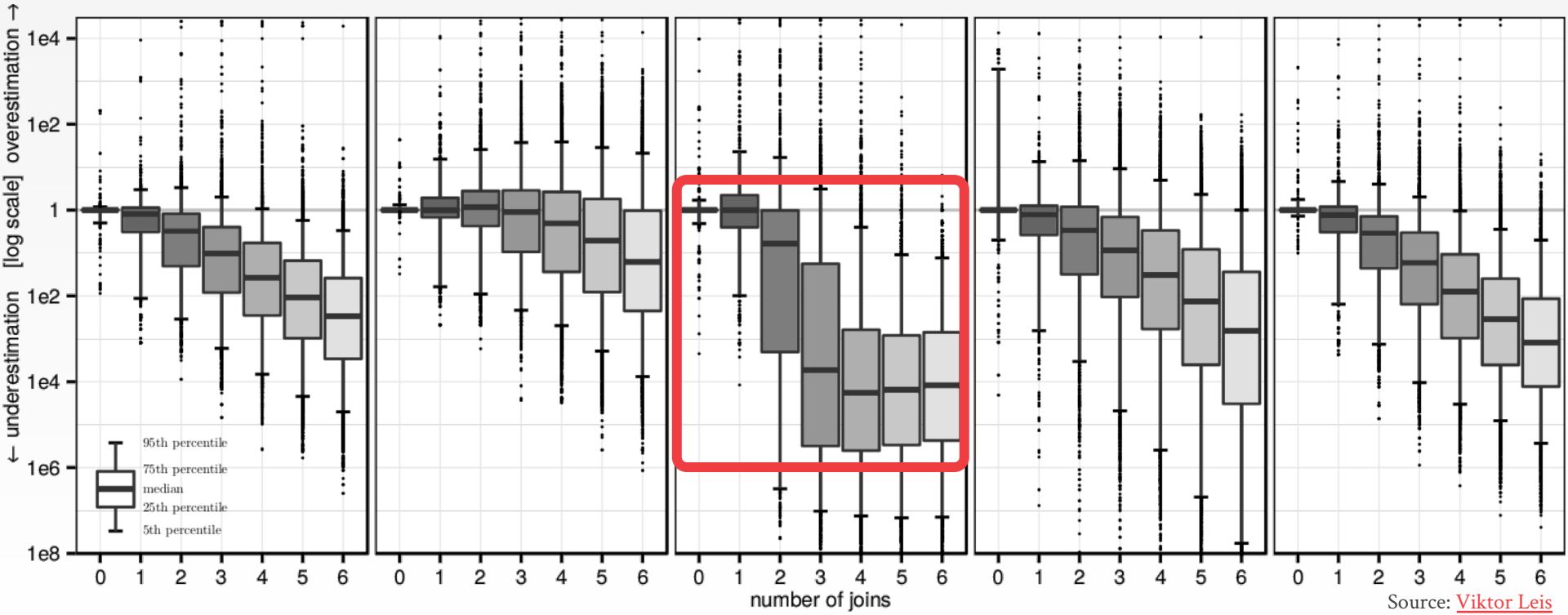
Source: Viktor Leis

ESTIMATOR QUALITY



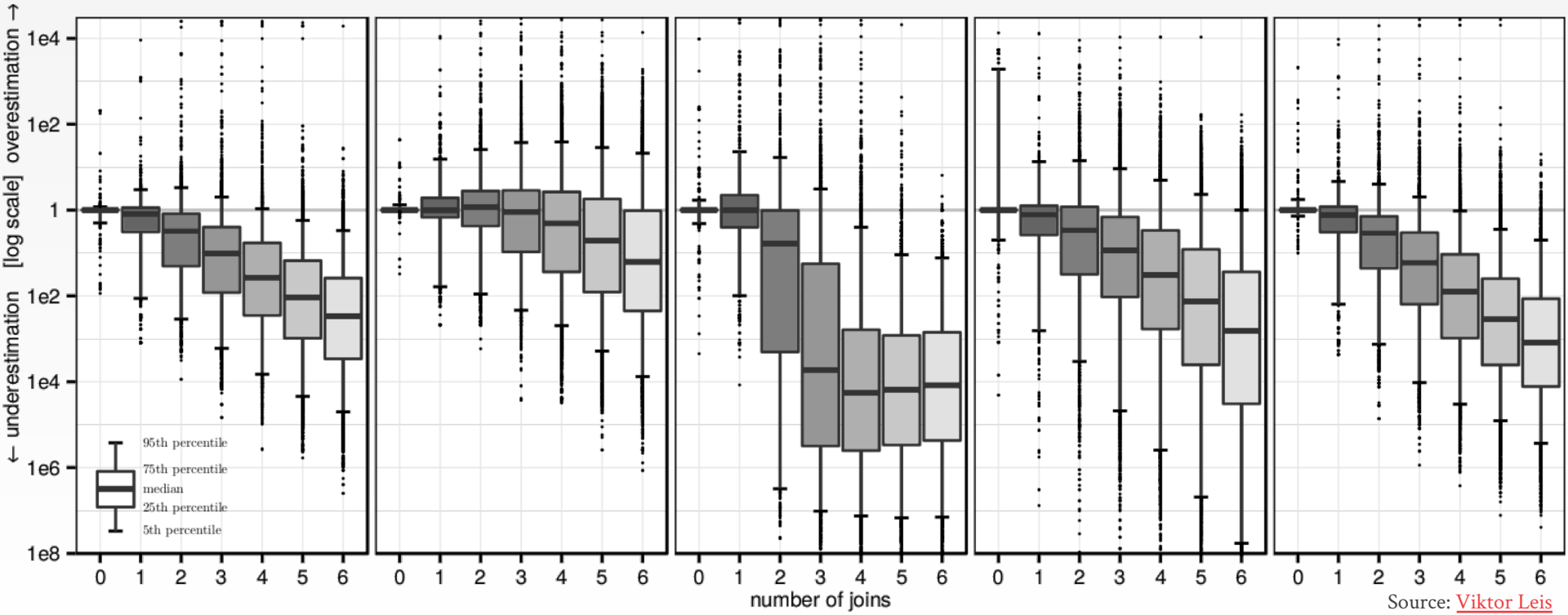
Source: Viktor Leis

ESTIMATOR QUALITY



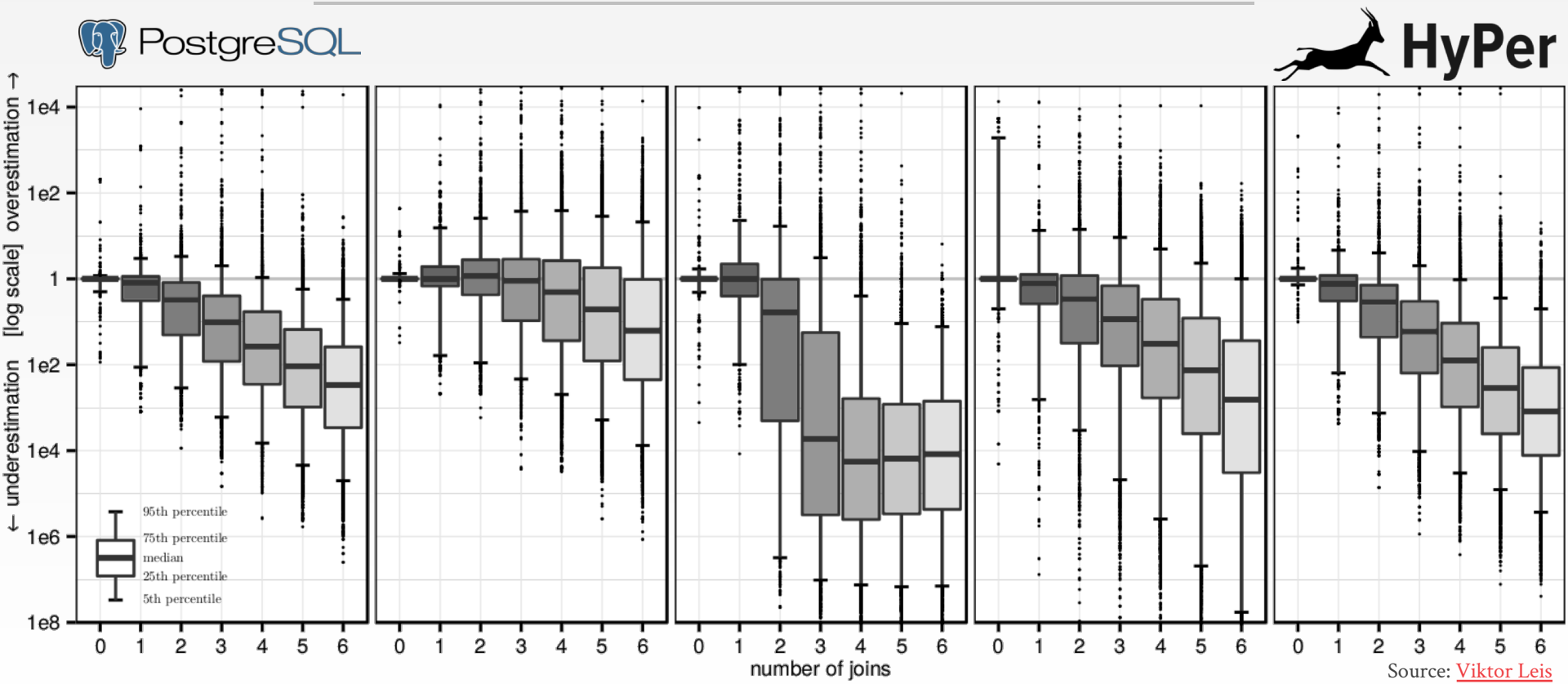
Source: Viktor Leis

ESTIMATOR QUALITY



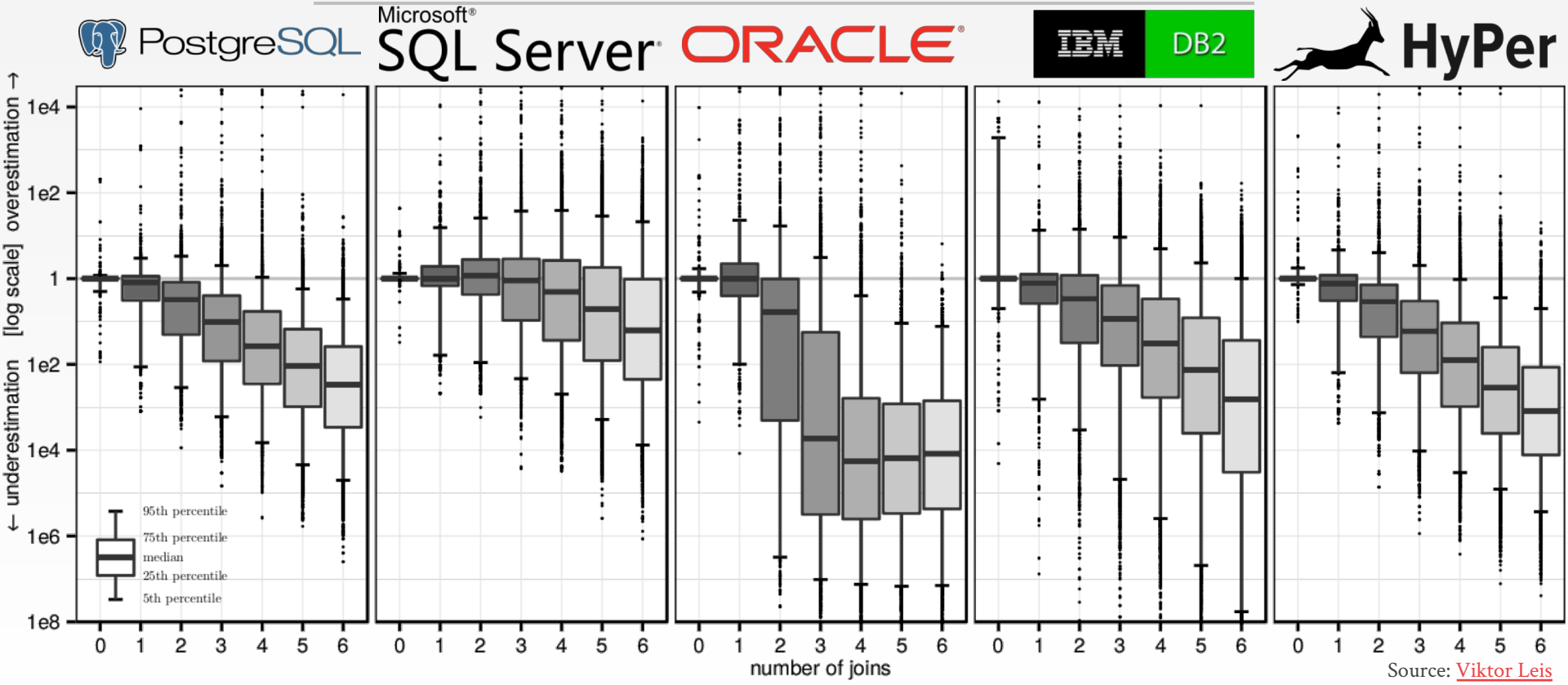
Source: Viktor Leis

ESTIMATOR QUALITY



Source: Viktor Leis

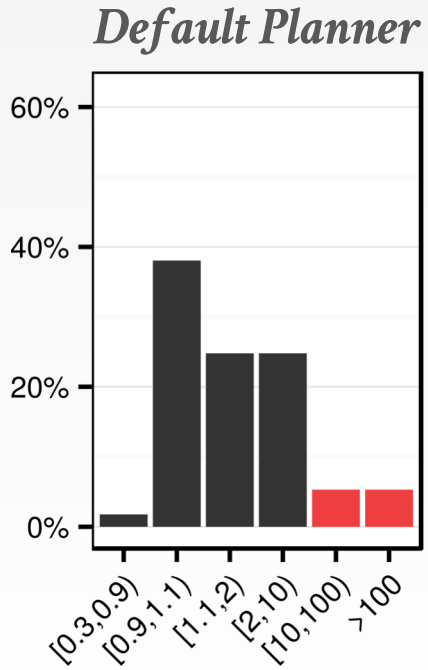
ESTIMATOR QUALITY



Source: Viktor Leis

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload

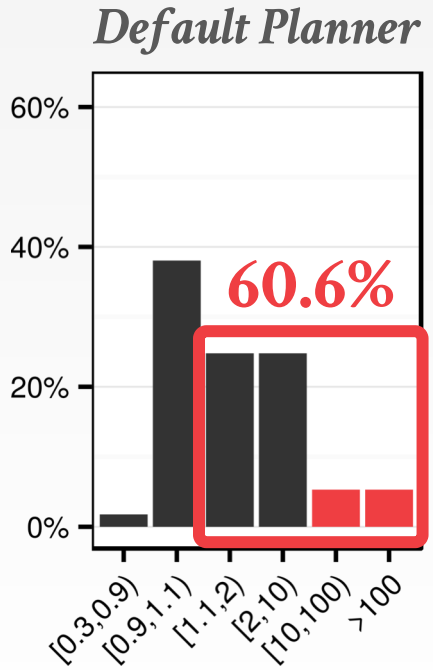


Slowdown compared to using true cardinalities

Source: Viktor Leis

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload

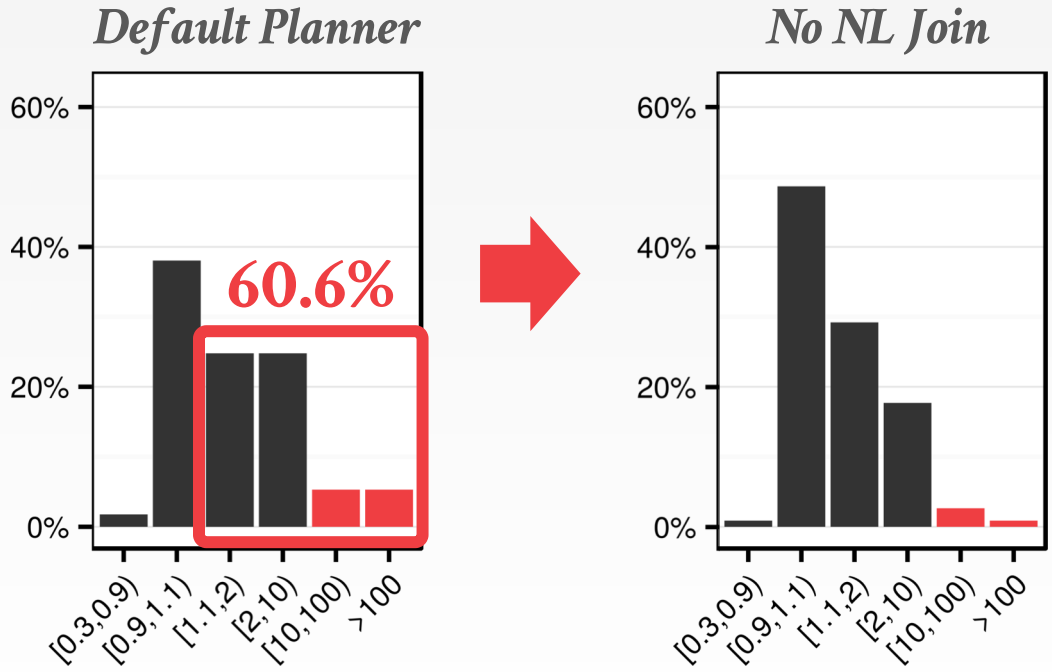


Slowdown compared to using true cardinalities

Source: Viktor Leis

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload

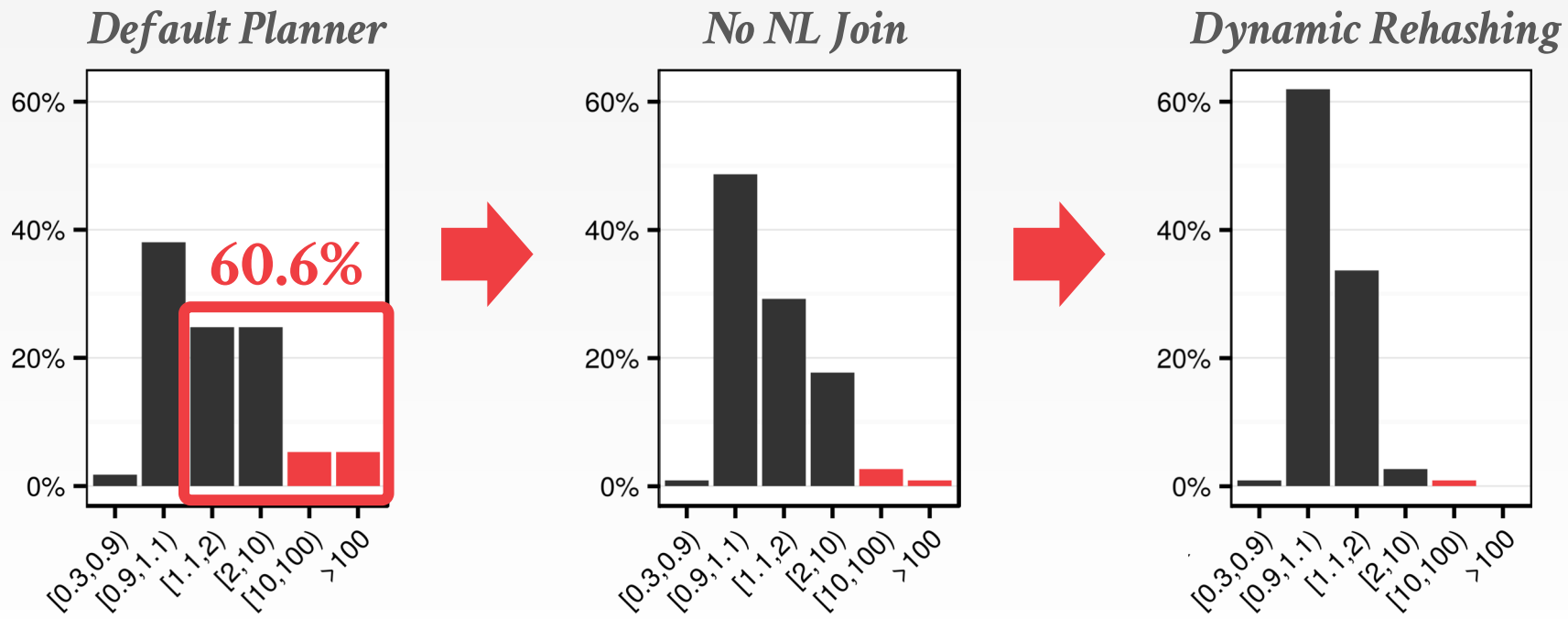


Slowdown compared to using true cardinalities

Source: Viktor Leis

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload



Slowdown compared to using true cardinalities

Source: Viktor Leis

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.

TIPS FOR TECHNICAL WRITING

TIPS FOR TECHNICAL WRITING

Technical writing is a balancing act between precision, clarity and marketing

→ Improving technical depth

→ Improving readability

Source: The Task of a Referee

IMPROVING TECHNICAL DEPTH

→ Problem Description
→ Significance
→ Relevance
→ Novelty
→ Validity
→ Contribution

PROBLEM DESCRIPTION

What is the problem being considered?

→ Is it clearly stated?

→ Do you make clear what the important issues are?

→ Do you tell, early in the paper, what you have accomplished?

→ For example, if this is a system description, has the system been implemented or is this just a design?

SIGNIFICANCE

Is the goal of this paper significant?

→ Is the problem real?

→ Is there any reason to care about the results of this paper, assuming for the moment that they are correct?

→ Is the problem or goal major, minor, trivial or non-existent?

RELEVANCE

Relevance

→ **Timeliness:** Is the problem now obsolete, such as reliability studies for vacuum tube mainframe computers?

→ **Specificity:** Is the problem so specific or so applied as to have no general applicability and thus not be worth wide publication?

NOVELTY

Is the problem, goal, or intended result new?

→ Has it been built before?

→ Has it been solved before?

→ Is this a trivial variation on or extension of previous results?

→ Are you aware of related and previous work, both recent and old?

VALIDITY

Is the method of approach valid?

→ What are the assumptions?
→ How realistic are they?
→ If they aren't realistic, does it matter?
→ How sensitive are the results to the assumptions?

CONTRIBUTION

What should the reader learn from this paper?

→ If you didn't learn anything, and/or if the intended reader won't learn anything, the paper is not publishable

IMPROVING READABILITY

→ Use bulleted lists
→ Remove salt & pepper words
→ Remove beholder words
→ Remove lazy words
→ Avoid adverbs
→ Paper strengths
→ Leverage tools

#1: USE BULLETED LISTS

Avoid verbose paragraphs

→ Use bulleted lists instead

→ Scope out the structure of the paper before expanding the bullet points

→ Example: First, second and third components of system

AVOID WEASEL WORDS

Weasel words obscure precision.

→ Phrases or words that sound good without conveying information.

→ Type 1: Salt & Pepper words

→ Type 2: Beholder words

→ Type 3: Lazy words

#2: REMOVE SALT & PEPPER WORDS

Students tend to sprinkle in salt and pepper words for seasoning.

→ These words look and feel like technical words, but convey nothing.
→ Examples: *various*, *a number of*, *fairly*, and *quite*.
→ Sentences that cut these words out become stronger.
→ Bad: It is quite difficult to find untainted samples.
→ Better: It is difficult to find untainted samples.
→ Bad: We used various methods to isolate four samples.
→ Better: We isolated four samples.

#3: REMOVE BEHOLDER WORDS

Beholder words are those whose meaning is a function of the reader

→ Example: *interestingly*, *surprisingly*, *remarkably*, or *clearly*.

→ Peer reviewers don't like judgments drawn for them.

→ Bad: False positives were surprisingly low.

→ Better: To our surprise, false positives were low.

→ Good: To our surprise, false positives were low (3%).

#4: REMOVE LAZY WORDS

Students insert lazy words in order to avoid making a quantitative characterization.

→ They give the impression that the author has not yet conducted said characterization. These words make the science feel unfirm and unfinished.

→ The two worst offenders in this category are the words *very* and *extremely*. Other offenders include *several, exceedingly, many, most, few, vast.*

→ Bad: There is very close match between the two semantics.

→ Better: There is a close match between the two semantics.

#5: AVOID ADVERBS

In technical writing, adverbs tend to come off as weasel words.

→ I'd even go so far as to say that the removal of all adverbs from any technical writing would be a net positive for my newest graduate students.

→ That is, students weaken a sentence when they insert adverbs more frequently than they strengthen it.

→ Bad: We offer a completely different formulation of QO.

→ Better: We offer a different formulation of QO.

#6: PAPER STRENGTHS

→ Bad: Open sourcing the algorithm.
→ Bad: Easy to implement algorithm using libraries.
→ Bad: Good job of describing optimizations at each step.
→ Bad: Paper also does a few real world tests.
→ Bad: Paper provides theoretical guarantees about the bounds.
→ Good: Detection of new, low-magnitude earthquakes that were previously not detected.
→ Good: Accelerates query processing by 100x.
→ Good: The authors consider human attributes such as limited cognitive load and short attention span.

#6: PAPER STRENGTHS

→ Bad: Since the authors collaborated with seismologists for their research, their domain knowledge is well represented.

→ Better: They introduce the following domain-specific optimizations: X, Y, Z.

#7: LEVERAGE TOOLS

Use a powerful typesetting system
→ A Very Short Introduction to LaTeX

Automate validation using writing tools
→ Shell scripts
→ Build paper: `make`
→ Check spelling: `make spellcheck`
→ Check style: `make stylecheck`
→ Grammarly

Source: Matt Might

NEXT CLASS

Query Execution & Scheduling!