Cardinality Estimation Done Right: Index-Based Join Sampling

Viktor Leis, Bernhard Radke, Andrey Gubichev,
Alfons Kemper, Thomas Neumann

Technische Universität München

Join Ordering

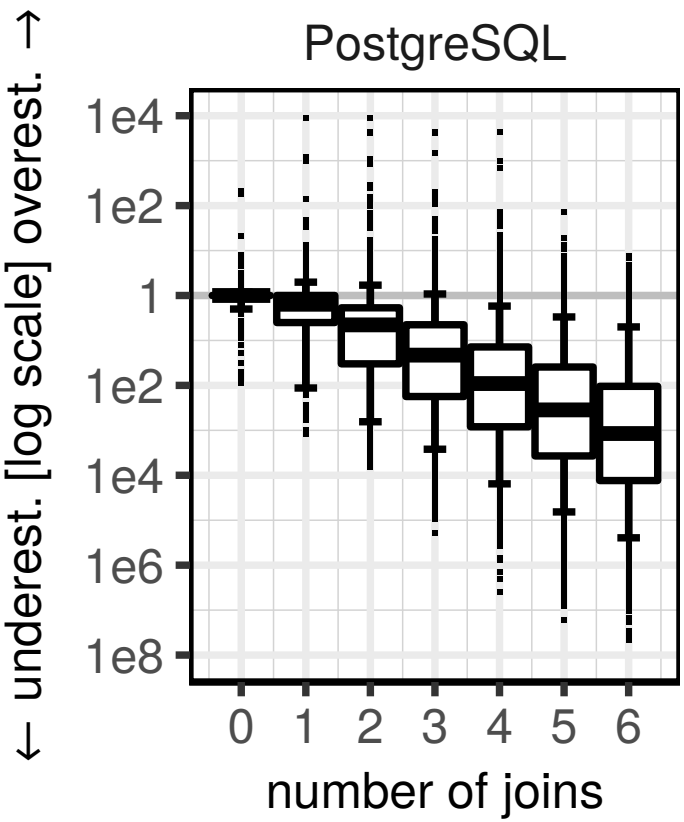
- finding a good join order is arguably the most important problem in query optimization
- the costs of different join orders often vary by orders of magnitude
- to distinguish good plans from bad ones, the cost of plans must be estimated using cardinality estimates for intermediate results
- example: to find the optimal join order for $A \bowtie_{A.id=B.aid} B \bowtie_{B.cid=C.id} C$, one needs estimates for $A \bowtie B$ and $B \bowtie C$

The State of the (Industrial) Art in Cardinality Estimation

- in virtually all systems estimation is based on:
  1. histograms and unique value counts
  2. strong assumptions: uniformity, independence, inclusion
- results with real-world data are terrible (Join Order Benchmark [Leis et al., VLDB 2016]):

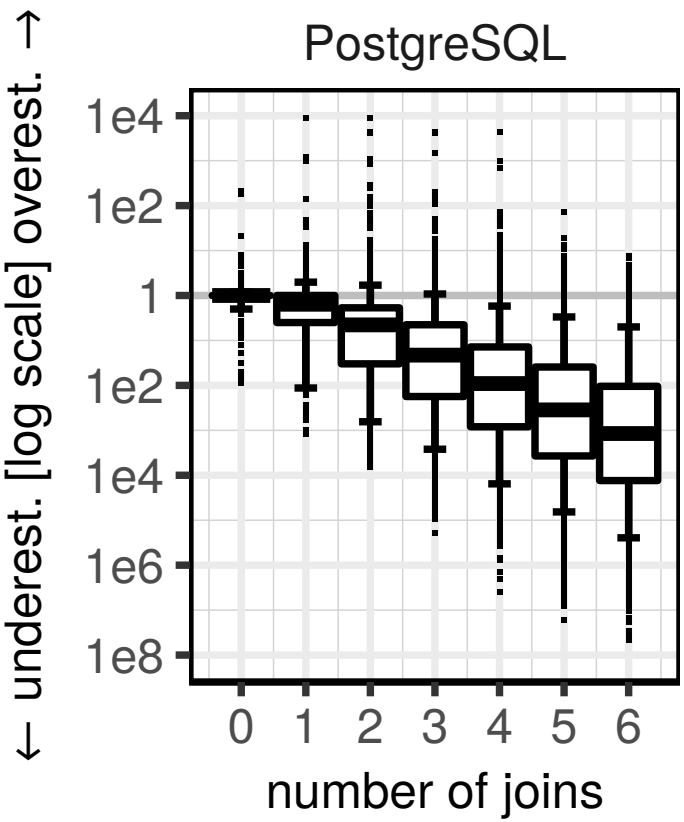
The State of the (Industrial) Art in Cardinality Estimation

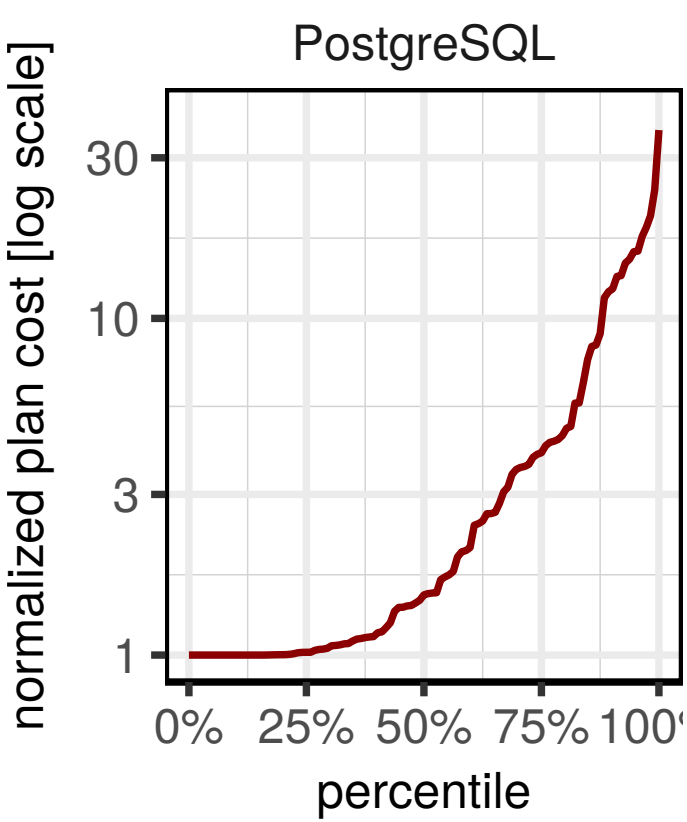
- in virtually all systems estimation is based on:
  1. histograms and unique value counts
  2. strong assumptions: uniformity, independence, inclusion
- results with real-world data are terrible (Join Order Benchmark [Leis et al., VLDB 2016]):



The State of the (Industrial) Art in Cardinality Estimation

- in virtually all systems estimation is based on:
  1. histograms and unique value counts
  2. strong assumptions: uniformity, independence, inclusion
- results with real-world data are terrible (Join Order Benchmark [Leis et al., VLDB 2016]):





Sampling to the Rescue?

- sampling, which does not rely on strong assumptions, is a highly promising alternative
- some systems (e.g., HyPer) use sampling to estimate base table selections:
  - keep random samples for each table (e.g., 10,000 rows) and execute selection on sample
  - produces accurate estimates for arbitrary predicates (correlations etc. are not a problem)
- How to use sampling for joins, which are the main source of errors?

Using Samples for Joins

We could pretend that samples are base tables and compute the join result

- would allows use to estimate the result size
- could become expensive, too (unlikely, but possible)
- usually, joining samples will produce small/empty results
- need a mechanism to sample the real join result
- but we cannot sample everything
- number of join candidates is exponential

Related (Sampling-Based) Work

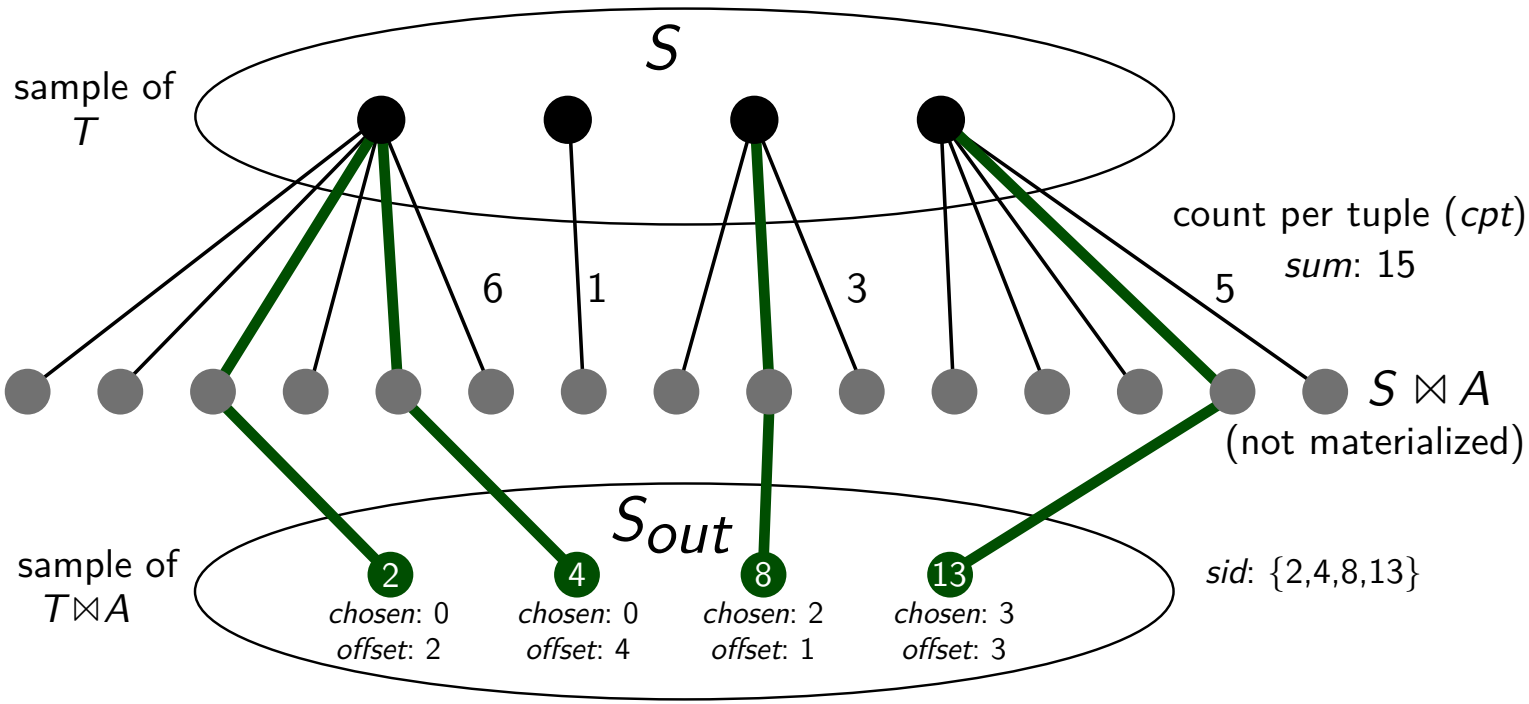
- *CS2* [Yu et al., SIGMOD 2013]
  - pre-materialize *correlated* samples to avoid joining independent samples
  - works well for star queries, but it is unknown how to apply this idea *automatically* for arbitrary queries
- *ROX* [Kader et al., SIGMOD 2009]
  - greedy heuristics that uses sampling through indexes to make more informed decisions
  - does not enumerate all join orders
- *Sampling-Based Re-Optimization* [Wu et al., SIGMOD 2016]
  - get plan from traditional optimizer
  - repeat until plan does not change: execute plan using 5% samples of each table
  - high overhead (large samples), avoids some bad plans but often misses optimal plan (no systematic exploration)
- sampling-based approaches proposed so far have weaknesses that preclude their use in industrial-strength systems

Index-based Join Sampling: Main Ideas

1. use existing index structures and fixed-size samples (1000) to get samples for larger intermediate results
2. systematically explore intermediate results in a bottom-up fashion (2-way joins, 3-way joins, . . .)
3. inject cardinalities computed in step 2 and run exhaustive join ordering algorithm (e.g., dynamic programming)

1. Cheap Sampling Using Indexes

- given a sample S obtain a sample for $S \bowtie_{S.id=A.id} A$ using an existing index on $A.id$:
  1. count the number of join partners for each tuple in S
  2. draw tuples at random to obtain desired number of results



- makes each sampling step cheap by avoiding "exploding" intermediate result sizes
- $O(|S|)$ or $O(|S| \log |A|)$, $|S| \leq 1000$

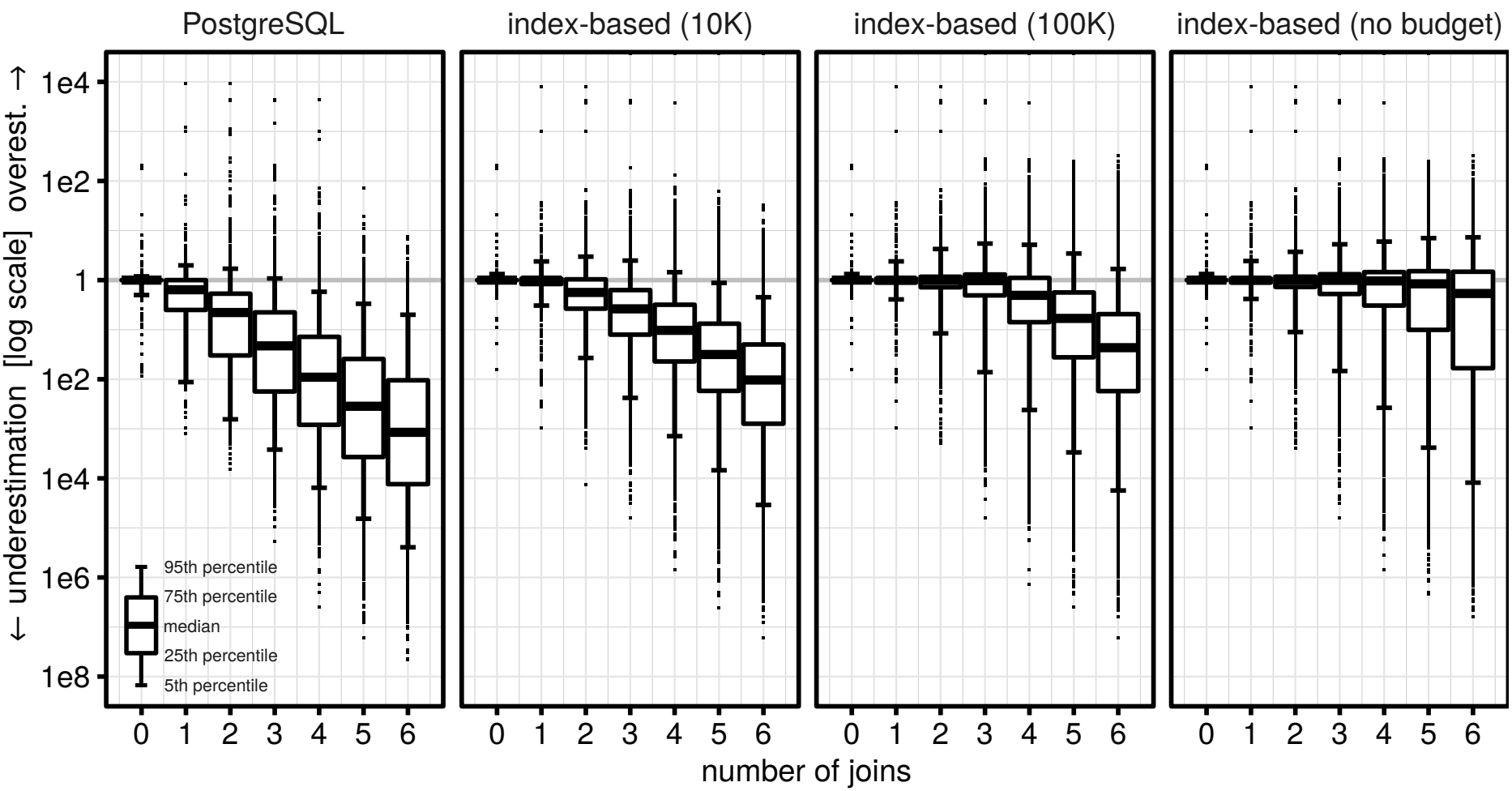
2. Systematic Bottom-Up Exploration

- generate samples for base tables
- join using existing indexes obtaining results for all 2-way joins, only then proceed to 3-way joins
- avoids "fleeing to ignorace" of the optimizer
- we need only one estimate per equivalence class
- stop early if sampling budget (e.g., 100K lookups) runs out
  - budget is a parameter that determines how much time is spent in the additional sampling phase

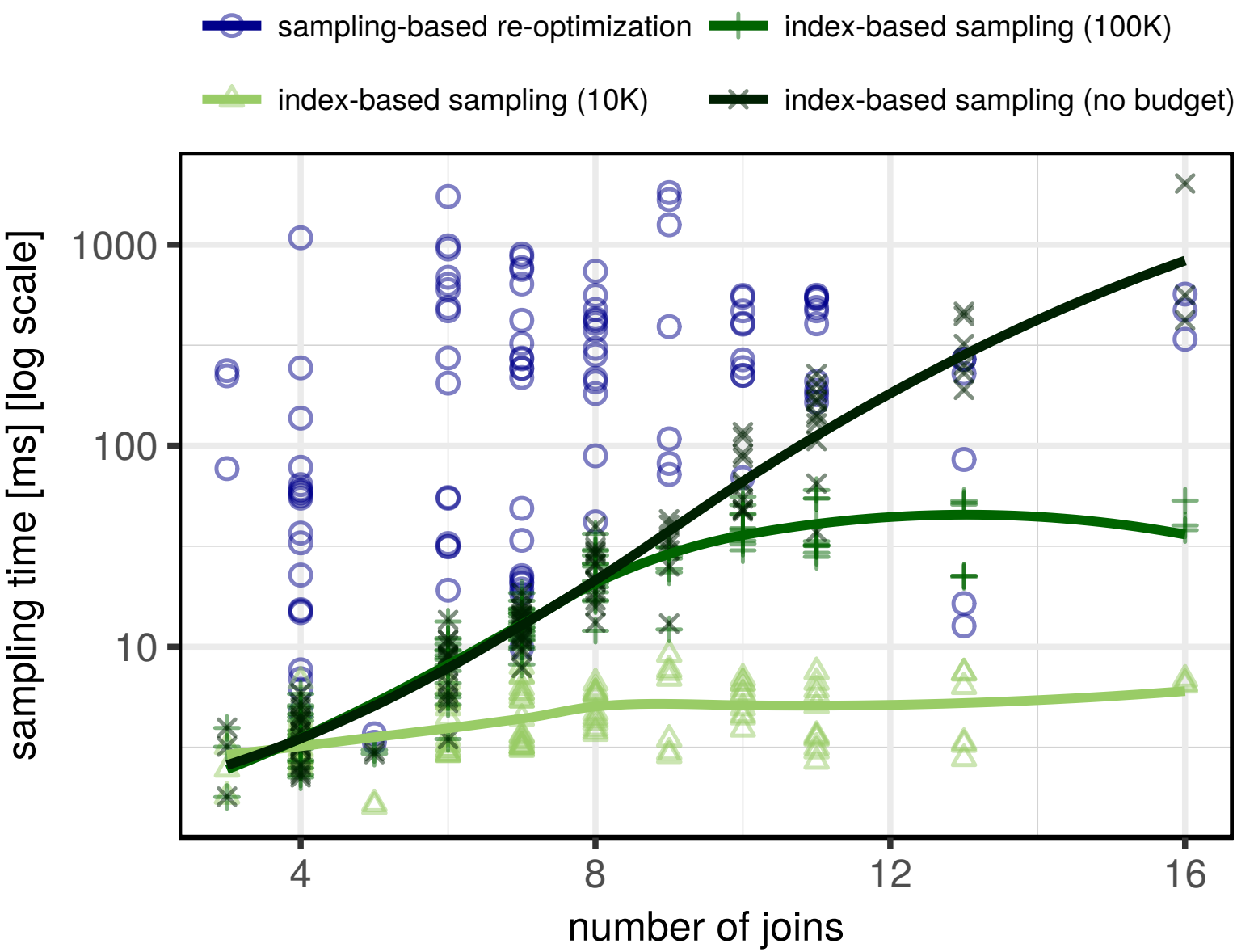
3. Join Ordering

- inject cardinalities from step 2 into traditional optimizer
  - fall back to traditional estimation if a result could not be estimated using sampling (due to missing indexes, a too small budget, or very high selectivities)
- run traditional (typically exhaustive) join enumeration algorithm, which now has much more accurate information
- execute resulting plan

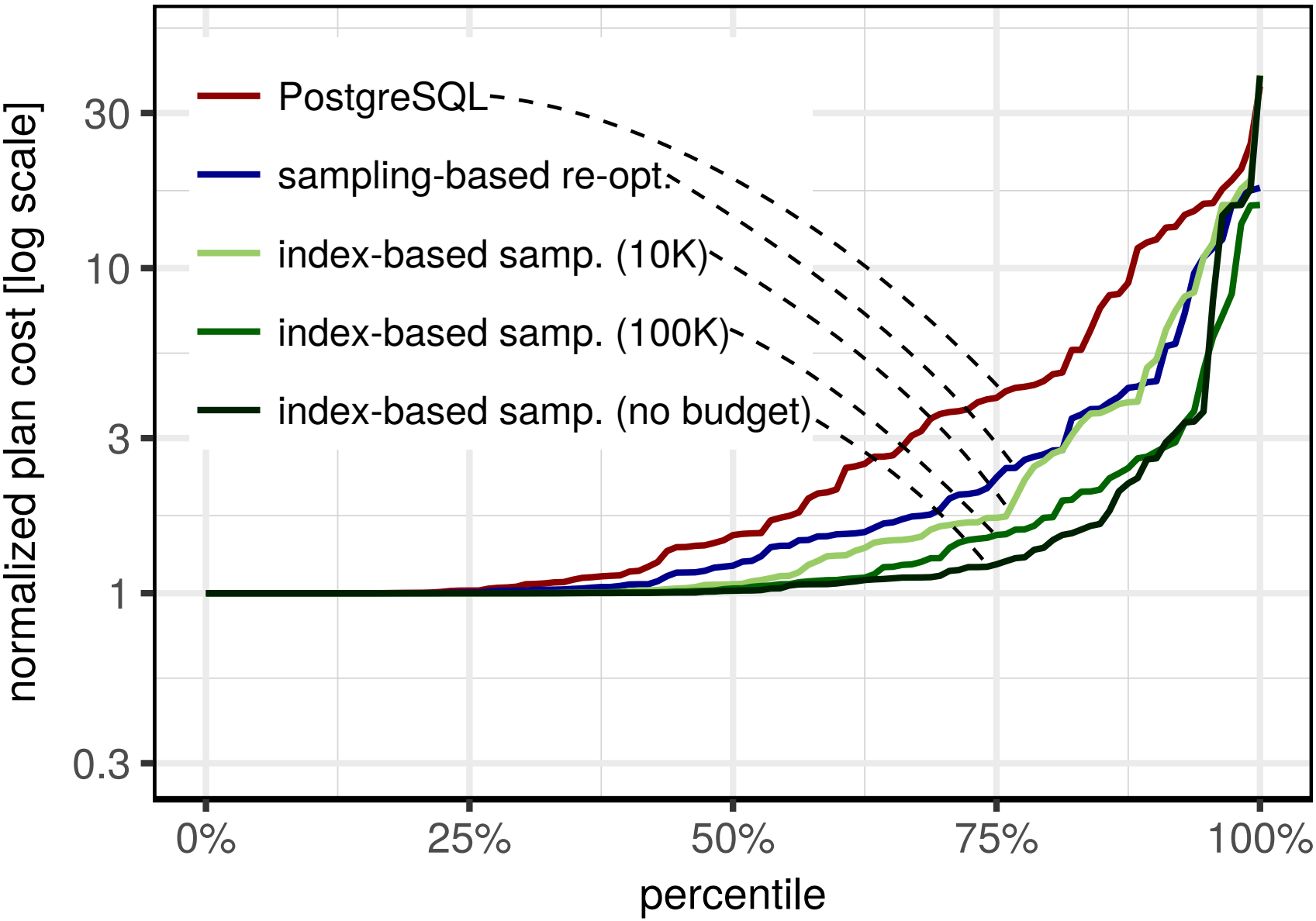
Does Sampling Improve Estimation?



How Expensive is Sampling?



Does Sampling Improve Plan Quality?



Summary

- index-based join sampling is an effective approach for cardinality estimation in main-memory database systems
- considerably improves estimation and plan quality
- low and configurable overhead
- easy to integrate into existing systems
- also possible as an optional phase (e.g., triggered by the user) for hard, long-running queries