DBEst: Revisiting Approximate Query Processing Engines with Machine Learning Models

Presented by Kehan Wang
Jinshu Yang

Authors

Qingzhi Ma
University of Warwick

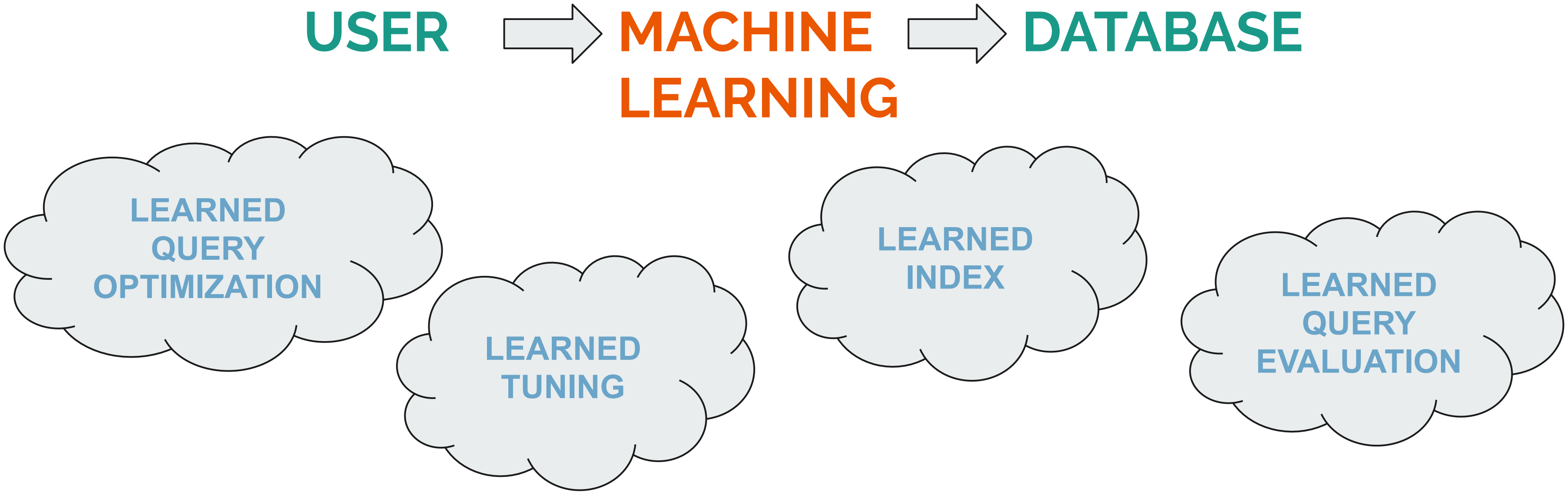
Peter Triantafillou
University of Warwick

Agenda

- Introduction
- AQP Engine
- DBEst Architecture
- DBEst Implementation
- Evaluation
- Related Work
- Conclusion & Future works

Introduction

Background


USER
MACHINE LEARNING
DATABASE
LEARNED QUERY OPTIMIZATION
LEARNED TUNING
LEARNED INDEX
LEARNED QUERY EVALUATION

What is the problem?

Selection Operators

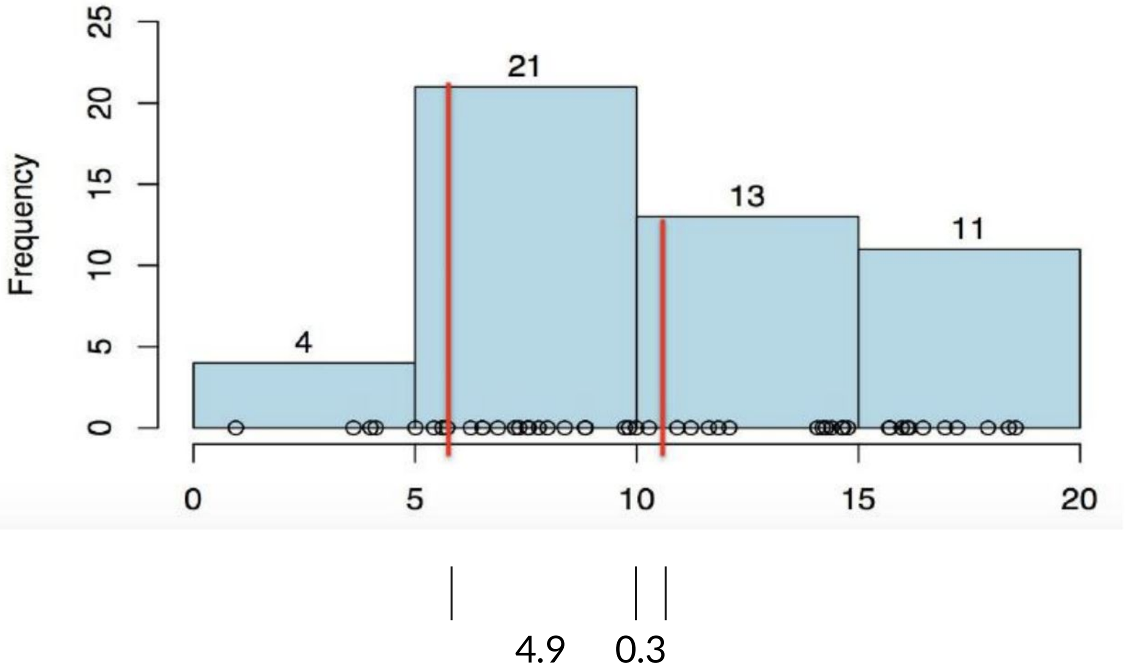
Aggregate Functions

```
SELECT  AF(y) FROM T
WHERE x BETWEEN lb AND ub
```

Limitations?

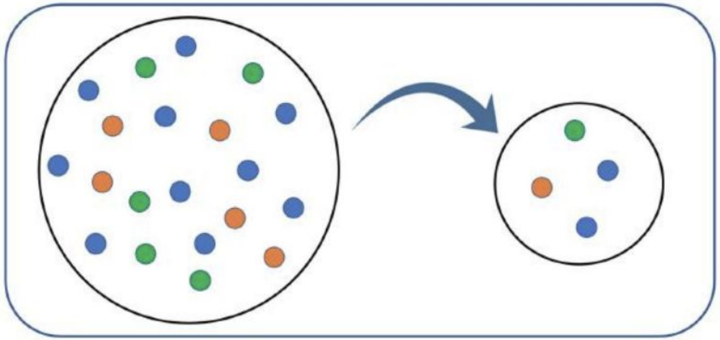
AQP (Approximate Query Processing)

Histogram:
- SELECT COUNT(x) WHERE 5.1 < x < 10.3
- Exact answer: 21
- Approximate answer:
  (4.9/5) * 21 + (0.3/5) * 13 = 21.36



AQP State of Art

SAMPLING



AQP-Sampling

	Offline Sampling	Online Sampling
Assumption	Workload is partially known.	No assumptions
SpeedUp	High	Low

AQP

Can we do AQP while ensuring: ?

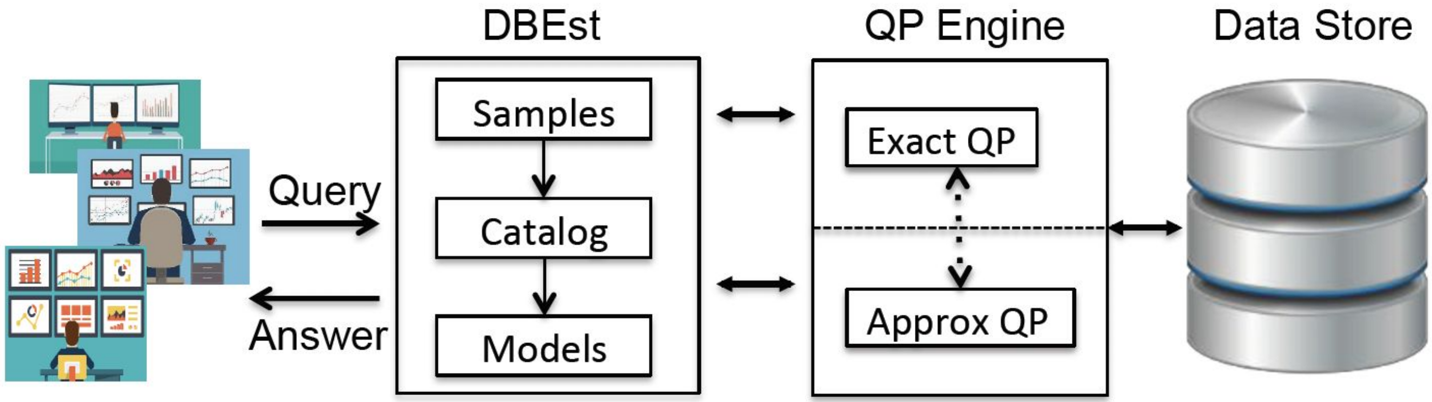
- Small query execution time
- Small states (memory/storage)
- High accuracy
- Low money cost

YES!

DBEst

DEFINITION:

an AQP engine supporting the analytical needs, using prebuilt, a priori state



DBEst-What are the models?

TWO MODELS

D(x)

Density Estimator

R(x)

Regression Function

DBEst-How?

SELECT COUNT(sales_price,x) FROM store_sales
WHERE (sales_price,x) BETWEEN lb AND ub;

$$COUNT(y) \approx N \cdot \int_{lb}^{ub} D(x)dx$$

DBEst-How?

$$VARIANCE\_x(x) = \mathbb{E}\left[x^2\right] - [\mathbb{E}[x]]^2$$

$$= \frac{\int_{lb}^{ub} x^2 D(x)dx}{\int_{lb}^{ub} D(x)dx} - \left[\frac{\int_{lb}^{ub} xD(x)dx}{\int_{lb}^{ub} D(x)dx}\right]^2$$

$$STDDEV\_x(x) = \sqrt{VARIANCE\_x(x)}$$

$$= \sqrt{\frac{\int_{lb}^{ub} x^2 D(x)dx}{\int_{lb}^{ub} D(x)dx} - \left[\frac{\int_{lb}^{ub} xD(x)dx}{\int_{lb}^{ub} D(x)dx}\right]^2}$$

DBEst-How?

PERCENTILE

$$P(x < \alpha) = p$$

$$\int_{-\infty}^{\alpha} D(x)dx = p$$

$$F(x) = p$$

USE BISECTION!

DBEst-How?

SELECT COUNT(x) FROM table
WHERE x BETWEEN lb AND ub;

SELECT SUM(y) FROM table
WHERE x BETWEEN lb AND ub;

Problem: we need y to query!

USE $R_y(x)$

DBEst-How?

SELECT AVG(sales_price) FROM store_sales
WHERE sales_time BETWEEN lb AND ub;

$$AVG(y) = \mathbb{E}[y] \approx \mathbb{E}[R(x)] = \frac{\int_{lb}^{ub} D(x)R(x)dx}{\int_{lb}^{ub} D(x)dx}$$

SELECT AVG(y) FROM table
WHERE x BETWEEN lb AND ub;

$$E[f(x)] = \frac{\int_{lb}^{ub} f(x)D(x)\,dx}{\int_{-lb}^{ub} D(x)\,dx}$$

DBEst-How?

EXTEND REGRESSION TO MULTIVAR:

SELECT SUM(sales_price) FROM
store_sales
WHERE sales_time BETWEEN lb AND ub
AND sold_time BETWEEN lb AND ub ;

$R_z(x, y)$

SELECT SUM(z) FROM
table
WHERE x BETWEEN lb AND ub
AND yBETWEEN lb AND ub ;

DBEst-How?

```
SELECT  z, AVG(y) FROM T
WHERE x BETWEEN lb AND ub
GROUP BY z;
```

SOLUTION:
Treat each z as having its own dataset to train model primitive on

DBEst-LIMITATIONS AND CHALLENGES

Models grow linearly with number of groups

*increase query processing time? - parallelizable

SOLUTION: create Model bundles to store model necessary for “High-cardinality” queries

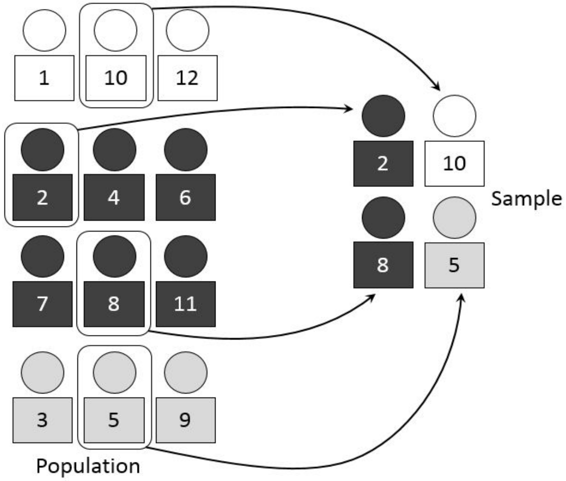
*still 10x as fast as sampling

DBEst Implementation

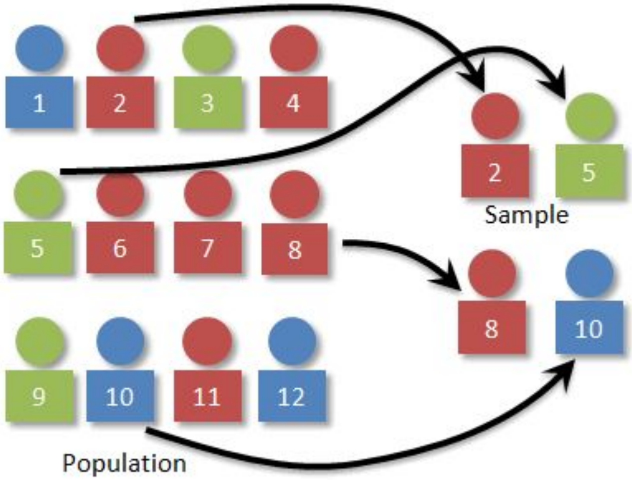
1. Sampling
2. Density Estimator
3. Regression Model Selection
4. Integral Evaluation
5. Parallel/Distributed Computation

Sampling

The paper mentions two sampling techniques



(i) Stratified sampling



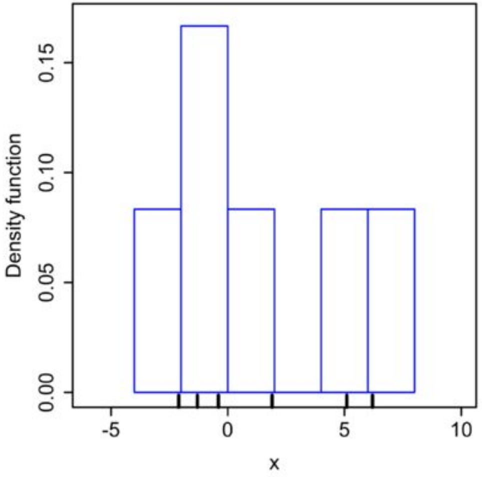
(ii) Reservoir Sampling

Why?

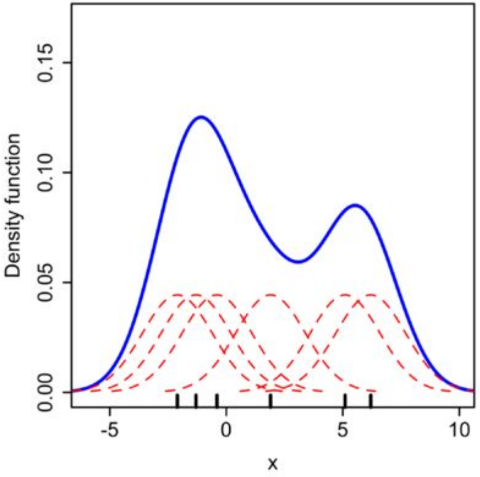
Density Estimator

- Kernel Estimator
    - High Accurate and Efficient
- Nearest neighbor method
- Orthogonal Series Estimators
- Histograms

Sample	1	2	3	4	5	6
Value	-2.1	-1.3	-0.4	1.9	5.1	6.2



Histograms



Kernel Estimator

Regression Model Selection

- LightGBM
- **XGBoost**
- GBoost

Someone may asking:
Select which models to build?

- Trying all combination for column sets
- Mining query logs (e.g BlinkDB)
- Depending on users (e.g VerdictDB)

Individual Model Training

Evaluate Accuracy for different models

Using GBoost Classifier to Select Best Model

Integral Evaluation

- Interesting accuracy-efficiency trade-offs!

$$I_w[lb,ub]f = \int_{lb}^{ub} w(x)f(x)dx \qquad I_w[lb,ub]f \approx \sum_{k=1}^{n} w_k f(x_k) \qquad \{R_{n_k}, E_{n_k}\}, k = 1, 2, \ldots, N$$

w(.) is a weight function

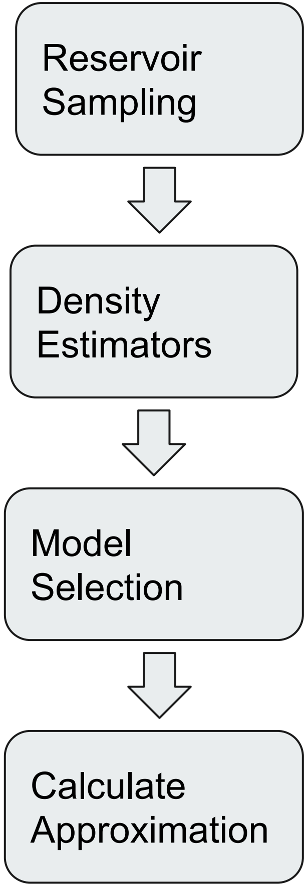
x1,x2,...,xn are nodes, and w1, w2, ...,
wn are weights

So what is happening here?

SELECT COUNT(pm25 real)

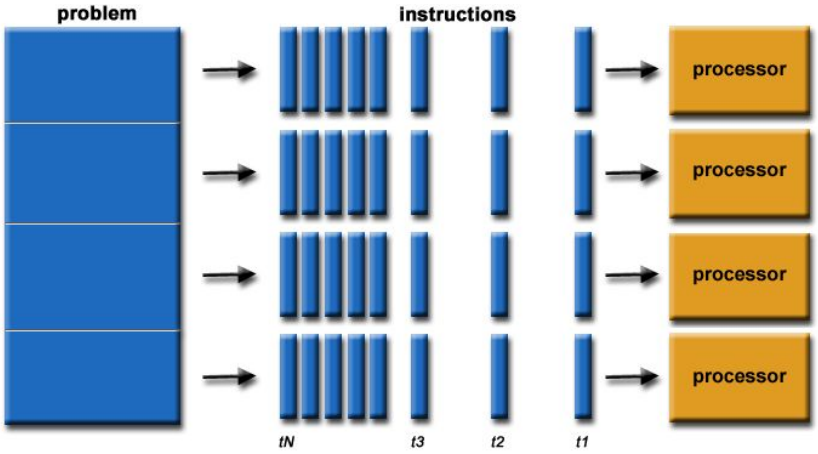
FROM mdl

WHERE PRES **BETWEEN** 1000 **AND** 1020;



Parallel/Distributed Computation

- Sampling -> easily parallelizable
  - Different nodes storing dataset partitions
- Model Training -> easily parallelizable
  - GROUP BY queries
- Query Processing -> easily parallelizable
  - Additional nodes/cores



Performance Evaluation

1. Experimental Setup: Ubuntu 18.04 with Intel Xeon X5650 12-core CPU, 64GB RAM and 4TB of SSD
2. Datasets: TPC-DS, Combined Cycle Power Plant(CCPP), Beijing PM2.5(UCI-ML)
3. Query Types:
   - Synthetic queries: w/ 0.1%, 1%, and 10% query range
   - Complex TPC-DS queries
4. Comparison:
   - w/ VerdictDB, Blink DB and MonetDB for error
   - w/ VerdictDB for time
5. Additional:
   - VerdictDB uses 12 cores while DBEst runs on 1 core(Multi-threaded DBEst is also evaluated)
   - Performance of joins and group by

Sensitivity Analysis (Query Range & Sample Size)

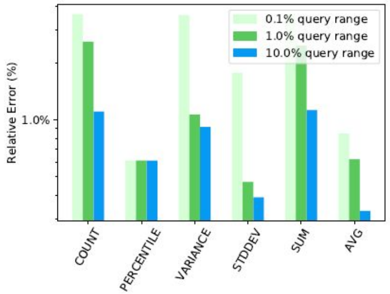


Figure 5: Influence of Query Range on Relative Error

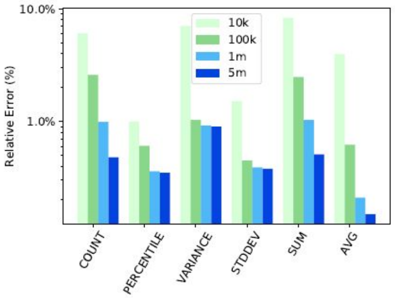
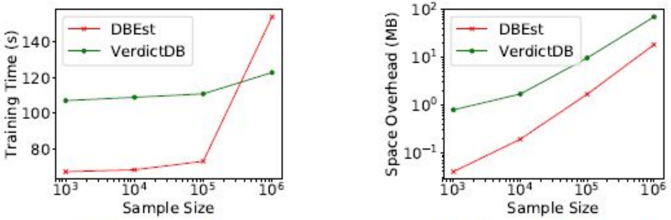


Figure 2: Influence of Sample Size on Relative Error



(a) Training Time

(b) Space Overhead

Figure 4: DBEst vs VerdictDB Overheads

- Dataset:TPC-DS
- Sample size: 100k rows
- Query Range: 0.1%, 1%, 10%

- Dataset:TPC-DS
- Query range: 1%
- Sample size: 10k, 100k, 1m, 5m

- 1 to 2 orders of magnitude less than VerdictDB's

Performance comparison TPC-DS & CCPP dataset

TPC-DS

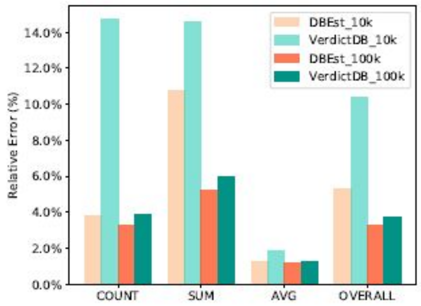


Figure 10: Relative Error: DBEst vs VerdictDB

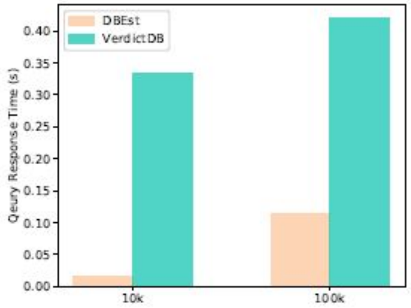
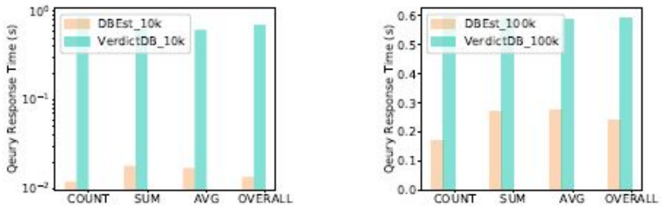


Figure 11: Response Time: DBEst vs VerdictDB

- Query range: 0.1%
- Sample size: 10k, 100k
- Response time is small

CCPP

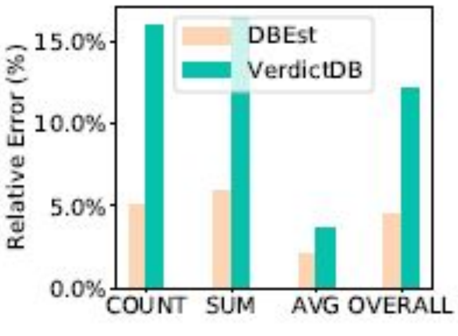


(a) 10k sample size (b) 100k sample size

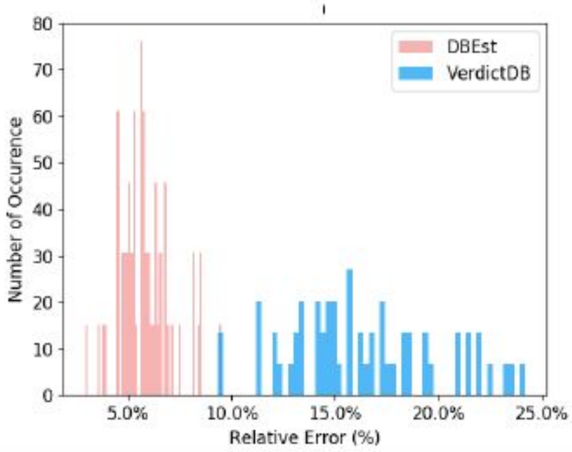
Figure 9: Response Time for CCPP Dataset

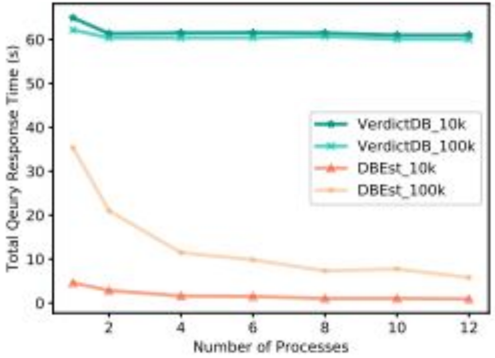
- Sample size: 10k, 100k
- Reminder: VerdictDB uses 12 cores, while DBEst uses 1 thread

Performance comparison GROUP BY



(a) Relative Error





- 90 queries, 57 groups
- Sample Size: 10k

- VerdictDB has no benefit from Parallel version

Performance comparison Join

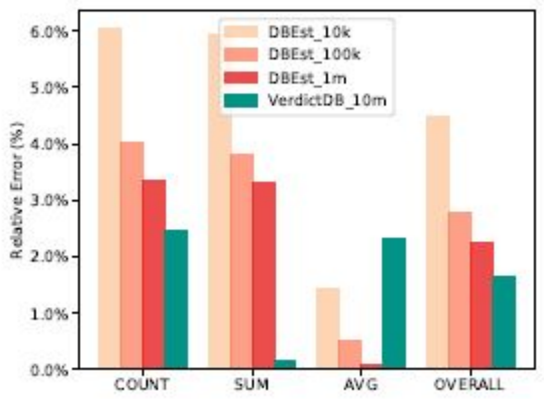
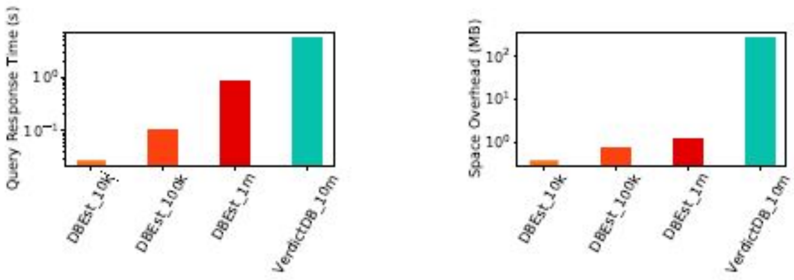


Figure 20: Join Accuracy Comparison



(a) Response Time

(b) Space Overhead

Figure 21: Join Performance Comparison

- Sample size: 10k, 100k, 1m for DBEst; 10m for VerdictDB

Limitation

- **GROUP BY queries:** As the number of group increase
  - Models ↑
  - Training time ↑
  - Query Response Time ↑
  - Space overheads ↑
- Small groups:
  - Building Models is an overkill
- No error guarantees

Conclusion & Future Works

- DBEst:
    - Smaller query response time
    - Higher accuracy
    - Smaller space-time overheads
    - scalability
    - All comes with low money cost!!
- Future:
    - Offering better efficiency-overheads-accuracy trade-offs(especially Joins queries)
    - categorical attributes
    - parallel/distributed DBEst