BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 24

Learned (Approximate) Query Processing

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

with slides from Marco Serafini and Peter Triantafillou

Project Submission & Presentations

April 27th, 11:59pm: *submit preliminary project report & code*

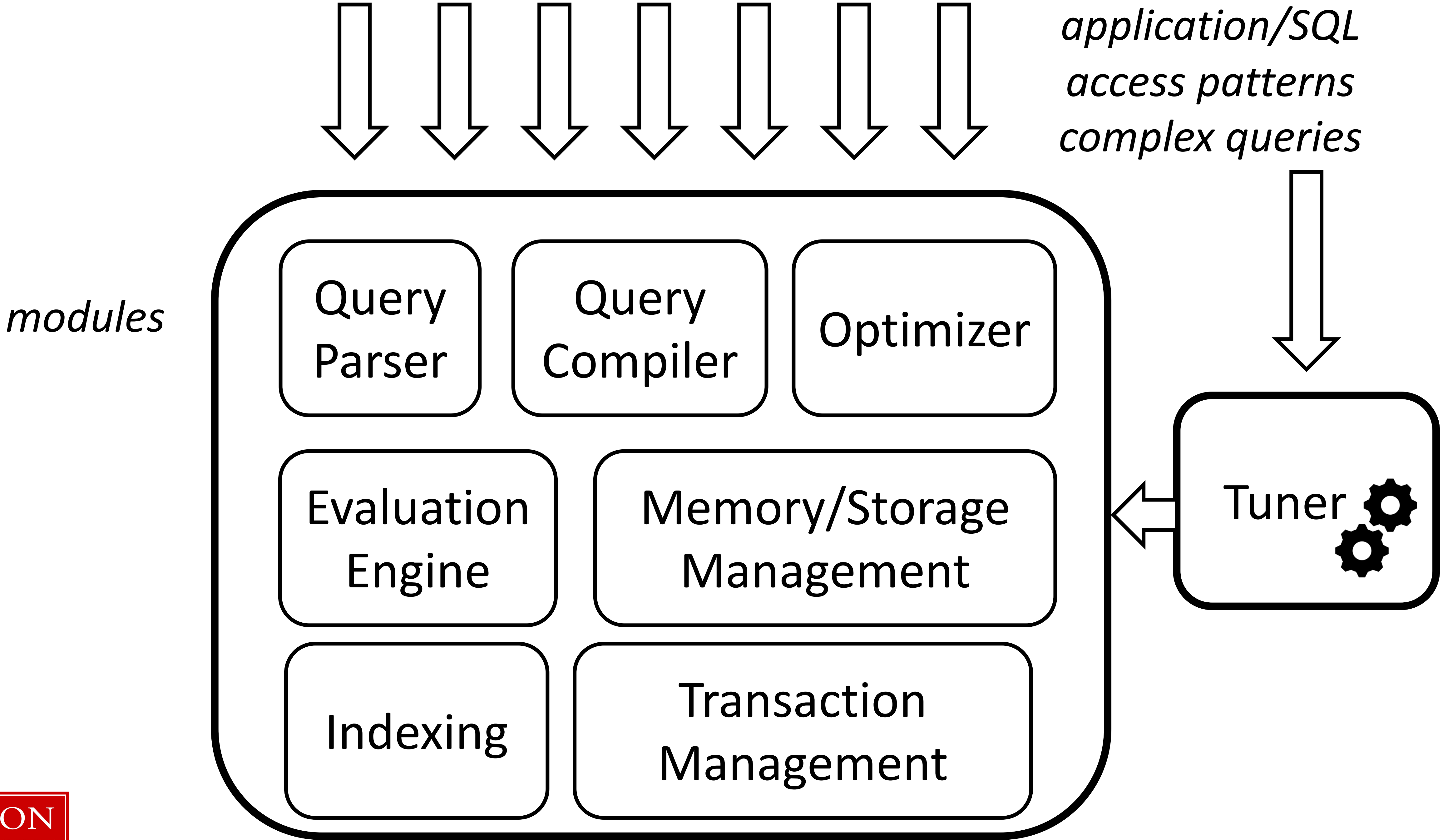
April 28th and May 3rd: *5 + 5 15-minute presentations (12+3 for questions)* (select your slot in piazza)

May 6th, 11:59pm (hard deadline): *send final report & updated code*

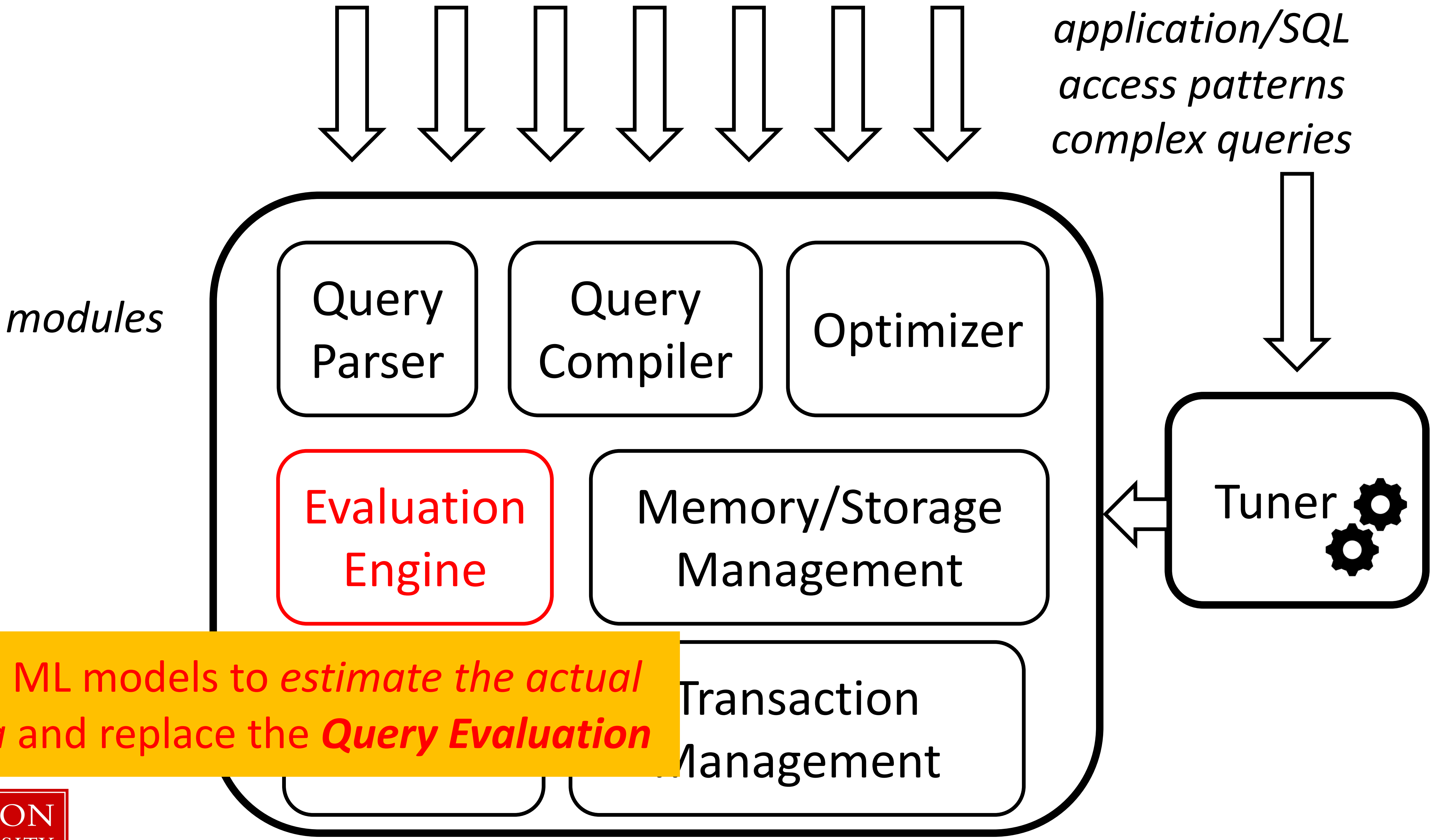
Guest lecture on *"Building a Healthcare Computational Engine: The case for purpose-built systems"*

Angelo Kastroulis, Ballista Technology Group

BOSTON UNIVERSITY



BOSTON UNIVERSITY



Use ML models to *estimate the actual data* and replace the ***Query Evaluation***

BOSTON UNIVERSITY

Motivation

In the era of big data, exact analytical query processing is too "expensive".

Agarwal, Sameer, et al. "BlinkDB: queries with bounded errors and bounded response times on very large data." *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013.

BOSTON UNIVERSITY

Motivation

In the era of big data, exact analytical query processing is too "expensive".

A large class of analytical queries takes the form:

SELECT AF(**y**) FROM table

WHERE **x** BETWEEN lb AND ub

[GROUP BY **z**]

Such queries are very popular on emerging datasets/workloads: IoT, sensors, scientific, etc.

Approximate Query Processing

Targeting ***Analytical*** Queries **– why?**

Goal: fast data analytics over large volumes of data

Tradeoff: accuracy vs. latency **– why?**

Is an accurate response always necessary?

exploratory analytics, business intelligence, analytics for ML

Basic tool: sampling

Current Solutions

- Online Aggregations
- Data Sketches
- Sample-based Approaches (the dominating approach)

Uniform Sampling

Stratified Sampling

Hash Sampling

- Limited supported aggregate functions
- Still, very time-consuming
- Space Overhead – samples can be very large
- Support for join (multi-way)
- Support for nesting

Query-time sampling

Queries ***explicitly specify*** sample operations

Sample then execute query

Uniform sampling: may miss small groups
Distinct sampler: online sampling of distinct values

With joins: want to sample ***before*** joins not after **– why?**

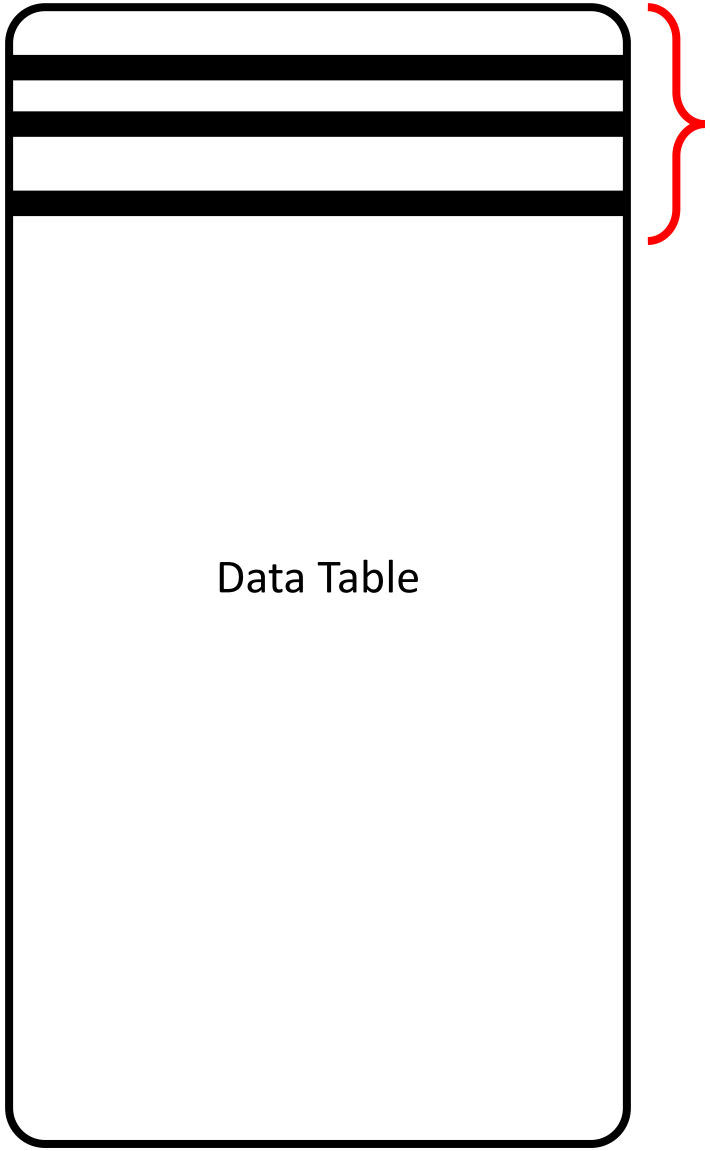
Online aggregation

Execute query on growing random samples

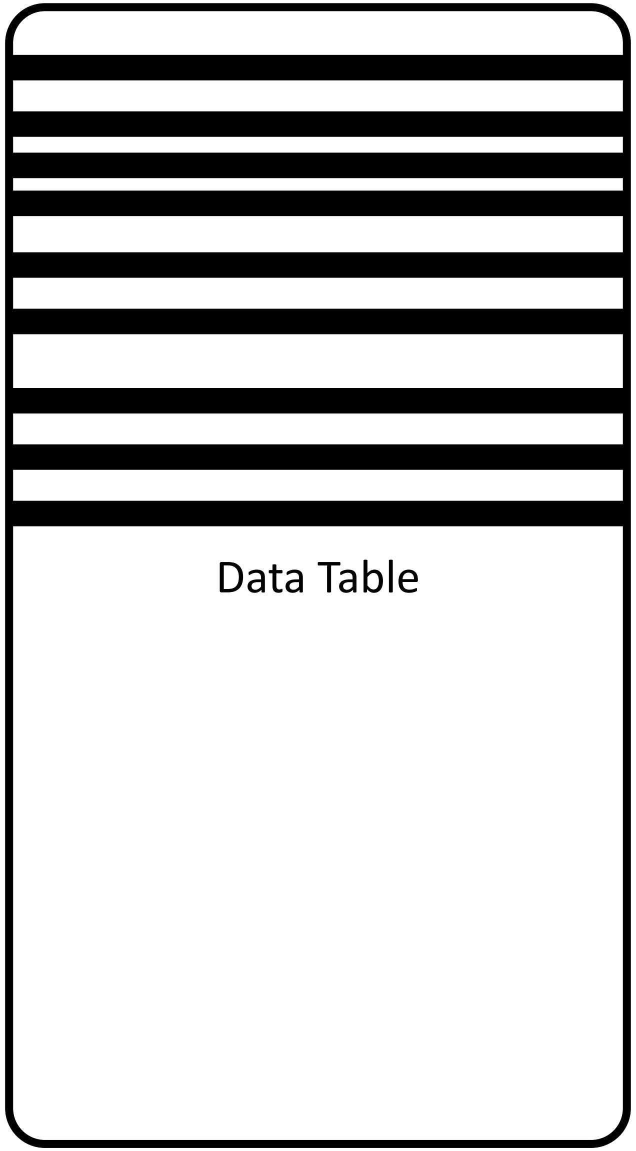
Preliminary outputs are constantly updated **– which?**

- Query result
- Estimated error

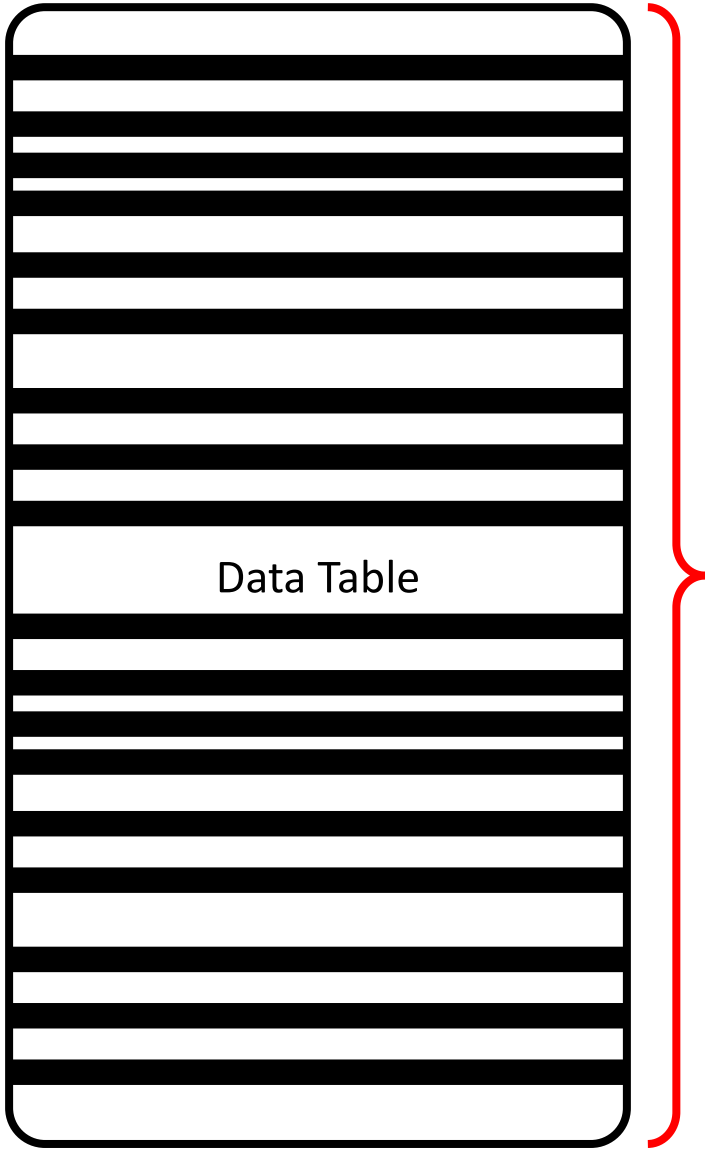
BOSTON UNIVERSITY



expected mean: 1003
[990, 1020] with confidence 95%



expected mean: 1002
[995, 1007] with confidence 96%



expected mean: 1001
[1001, 1001] with confidence 100%

Online aggregation

Execute query on growing random samples

Preliminary outputs are constantly updated **– which?**

- Query result
- Estimated error

Hard to execute efficiently **– why?**

- Random sample → Random access
- Random samples might contain few rows that join
- Can be improved using join indices

Queries on Pre-Computed Samples

Low latency because ***sampling cost*** is assumed ***offline***

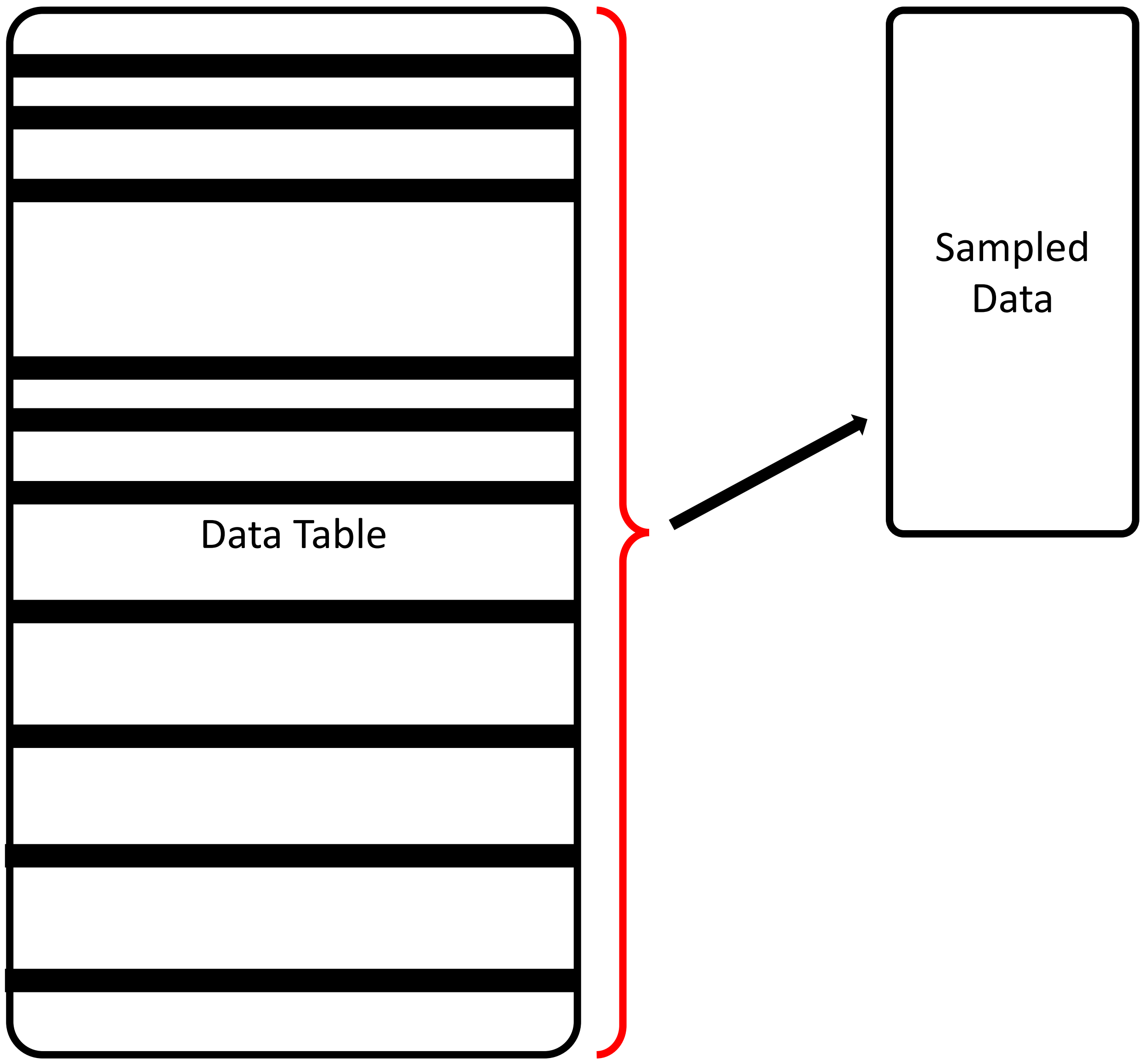
operate ***only on the sample***

Additional space (to keep sample)

Cannot provide fixed error bounds

Error bounds are data dependent (high variance = large error)

They can be arbitrarily large


Data Table
Sampled
Data

SQL additions

Aggregate is computed on a group

Group is defined based on certain columns

Extend specification with bounds

Error-bound query

```
SELECT count(*)
FROM Sessions
WHERE Genre=`western`
GROUP BY OS
ERROR WITHIN 10% AT CONFIDENCE 95%
```

Time-bound query

```
SELECT count(*)
FROM Sessions
WHERE Genre=`western`
GROUP BY OS
WITHIN 5 SECONDS
```

Offline vs online sampling

	Offline	Online
Assumption:	(partially) known workload	No assumption
Speedup:	High	Low

Offline vs online sampling

	Offline	Online
Assumption:	(partially) known workload	No assumption
Speedup:	High	Low

Offline vs online sampling

	Offline	**Online**
Assumption:	(partially) known workload	No assumption
Speedup:	High	Low

Both are helpful:

- offline sampling is used for (partially) predictable workloads,
- online sampling is for the rest.

DBEst: transparent AQP

Very small query <u>execution</u> times (e.g., ms),

With **small state** (memory/storage footprint) (e.g., KBs), and

High accuracy (e.g., a few % relative error)

Regardless *of data* ***size****?*

YES! (for a large class of analytical queries)

- rests on simple SML models
- Built over samples of tables

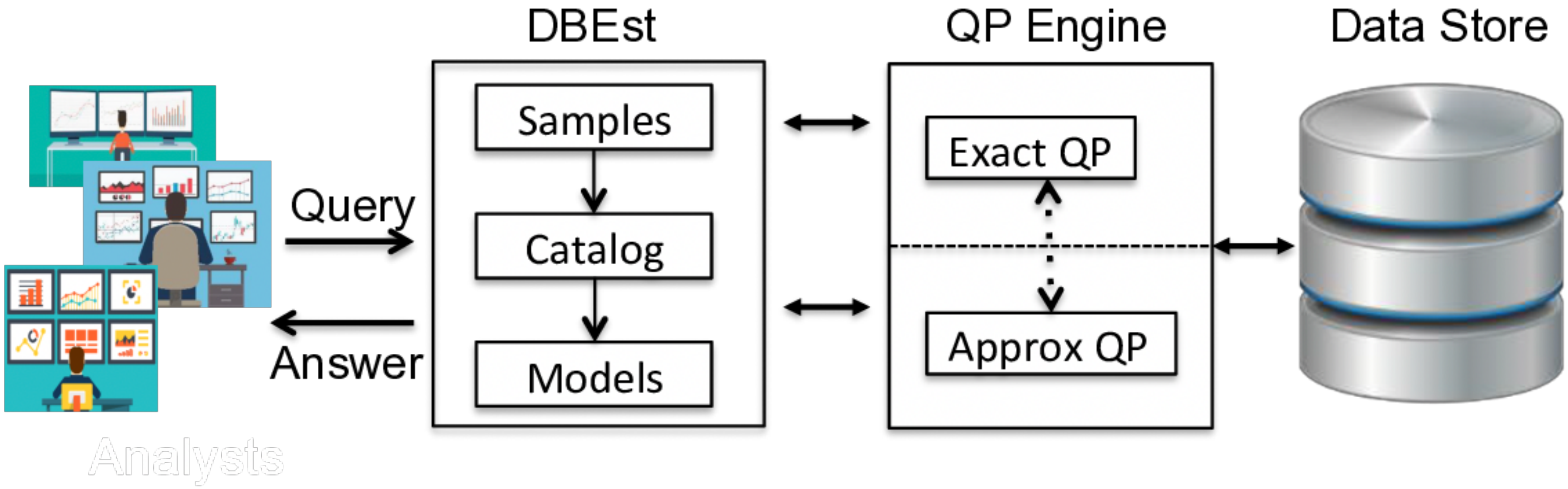
BOSTON UNIVERSITY

DBEst Contributions

DBEst shows that

- Models can be built over small samples
- Can generalize nicely, ensuring accuracy
- Model state is small (KBs)
- ***AQP over models is much faster than over samples***
- Model training overhead is acceptable – inline with sample generation.

DBEst Architecture



DBEst and ML models

*which **aggregate functions** are very **hard to answer** via **approximate** query processing?*

which are easy?

- Problem SQL query

  SELECT AF(**y**) from table

  WHERE **x** between *low* and *high*

  [GROUP BY **z**]

- What models?

Regression y=R(x)
- LR, PR...
- **XGBoost**, GBoost...

Density Estimator D(x)
- **Kernel Density**
- Nearest neighbor method
- Orthogonal series estimator

Density Estimator

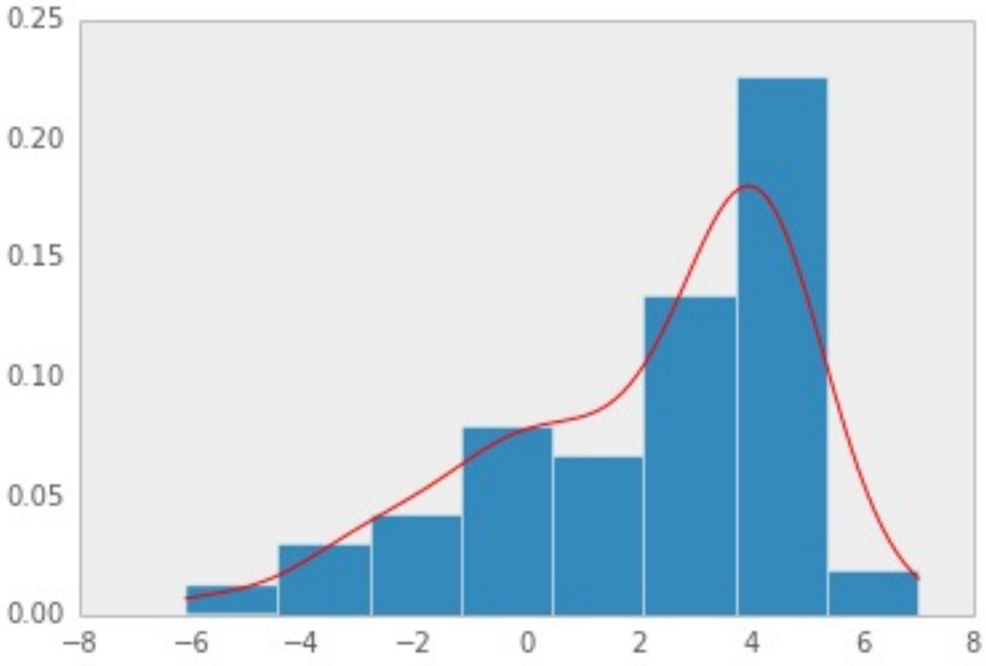


*image from wikipedia

Histograms is the simplest form of **density estimator**

DBEst is **gradually learning** a function
that **approximates** the **actual density** function of the data

e.g., "how many values exist between *low* and *hi*?"

BOSTON UNIVERSITY

Regression Model

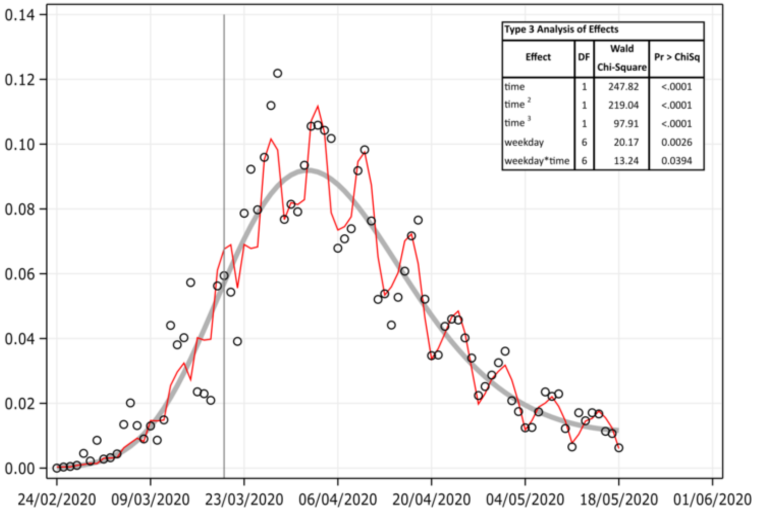


*image from wikipedia

A **regression model** describes the **relationship between two variables**
$y = F(x)$

DBEst uses a regression model to capture "matches" from selection

e.g., "which values of y exist for x between low and hi?"

How to use regression and density estimation to answer queries?

```
SELECT count(*)
FROM Table
WHERE x between lb and ub
```

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

fraction of values in [lb,ub]

```
SELECT avg(y)
FROM Table
WHERE x between lb and ub
```

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

relationship of x values with y values

fraction of values in [lb,ub]

```
SELECT sum(y)
FROM Table
WHERE x between lb and ub
```

$$\begin{aligned} SUM(y) &= COUNT(y) \cdot AVG(y) \\ &\approx COUNT(y) \cdot \mathbb{E}[R(x)] \\ &= N \cdot \int_{lb}^{ub} D(x)dx \cdot \frac{\int_{lb}^{ub} D(x)R(x)dx}{\int_{lb}^{ub} D(x)dx} \\ &= N \cdot \int_{lb}^{ub} D(x)R(x)dx \end{aligned}$$

BOSTON UNIVERSITY

How to use regression and density estimation to answer queries?

```
SELECT variance(y)
FROM Table
WHERE x between lb and ub
```

$$VARIANCE\_y(y) = \mathbb{E}\left[y^2\right] - \left[\mathbb{E}\left[y\right]\right]^2$$
$$\approx \mathbb{E}\left[R^2(x)\right] - \left[\mathbb{E}\left[R(x)\right]\right]^2$$
$$= \frac{\int_{lb}^{ub} R^2(x)D(x)dx}{\int_{lb}^{ub} D(x)dx} - \left[\frac{\int_{lb}^{ub} R(x)D(x)dx}{\int_{lb}^{ub} D(x)dx}\right]^2$$

```
SELECT percentile(x,p)
FROM Table
```

PERCENTILE.

If the reverse of the CDF, $F^{-1}(p)$, could be obtained, then the p^{th} percentile for Column x is

$$\alpha = F^{-1}(p) \tag{5}$$

Note that $F^{-1}(p)$ is derived using $\mathrm{F(p)} = \int_{-inf}^{p} D(x)dx$

BOSTON UNIVERSITY

More support on SQL

```
SELECT avg(y)
FROM Table
WHERE x1 between lb1 and ub1
  AND x2 between lb2 and ub2
```

$$
\begin{aligned}
AVG(y) &= \mathbb{E}\left[y\right] \\
&\approx \mathbb{E}\left[R(x_1, x_2)\right] \\
&= \frac{\int_{lb_1}^{ub_1} \int_{lb2}^{ub2} D(x_1, x_2) R(x_1, x_2) dx_2 dx_1}{\int_{lb_1}^{ub_1} \int_{lb_2}^{ub_2} D(x_1, x_2) dx_2 dx_1}
\end{aligned}
$$

Supporting GROUP BY

- build models for each group by value,
- create **model bundles**:
  - E.g., each bundle stores ~500 groups
  - Store bundles in, say, an SSD (~100 ms to deserialize and compute AF on bundle).

Supporting join

- Join table is flattened -> make samples -> build models.

Evaluation

systematically showing sensitivities on

- range predicate selectivity + sample sizes + AFs

Performance under Group By and Joins

Comparisons against

- State of the art AQP (VerdictDB and BlinkDB)
- State of the art columnar DB (MonetDB)

Using data from TPC-DS and 3 different UCI-ML repo datasets.

Experimental Setup

Ubuntu 18.04 with Xenon X5650 12-core CPU, 64 GB RAM And 4TB SSD

Datasets: TPC-DS, Combined Cycle Power Plant (CCPP), Beijing PM2.5

Query types:

- Synthetic queries: 0.1%, 1%, to 10% query range
- Number of queries: vary between 30 to1000 queries.
- Complex TPC-DS queries: Query 5, 7, and 77.

Compared against VerdictDB, BlinkDB and MonetDB, for error

- VerdictDB uses 12 cores while DBEst runs on 1 core. (Multi-threaded DBEst is also evaluated)

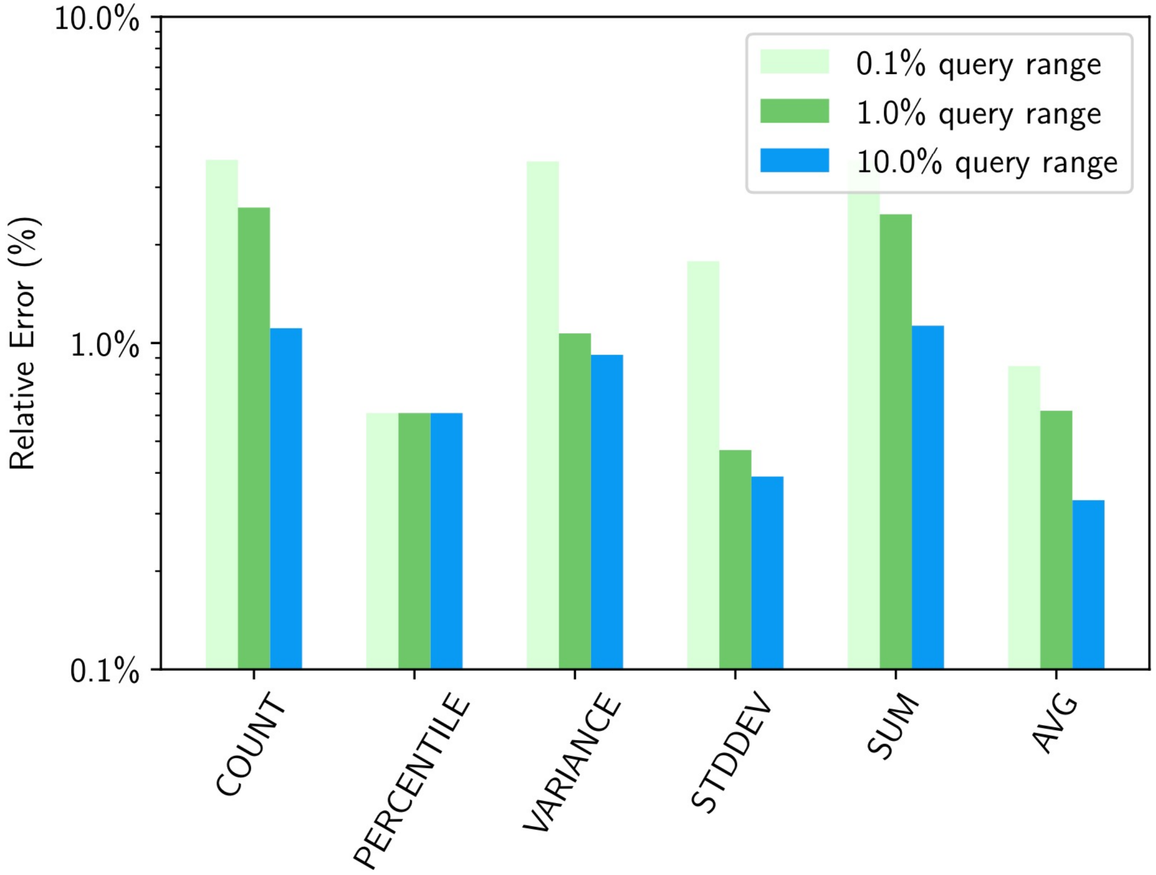
Report execution times + system throughput for the parallel version

Report performance of joins and group by

BOSTON UNIVERSITY

Performance – Sensitivity Analysis
Query range effect

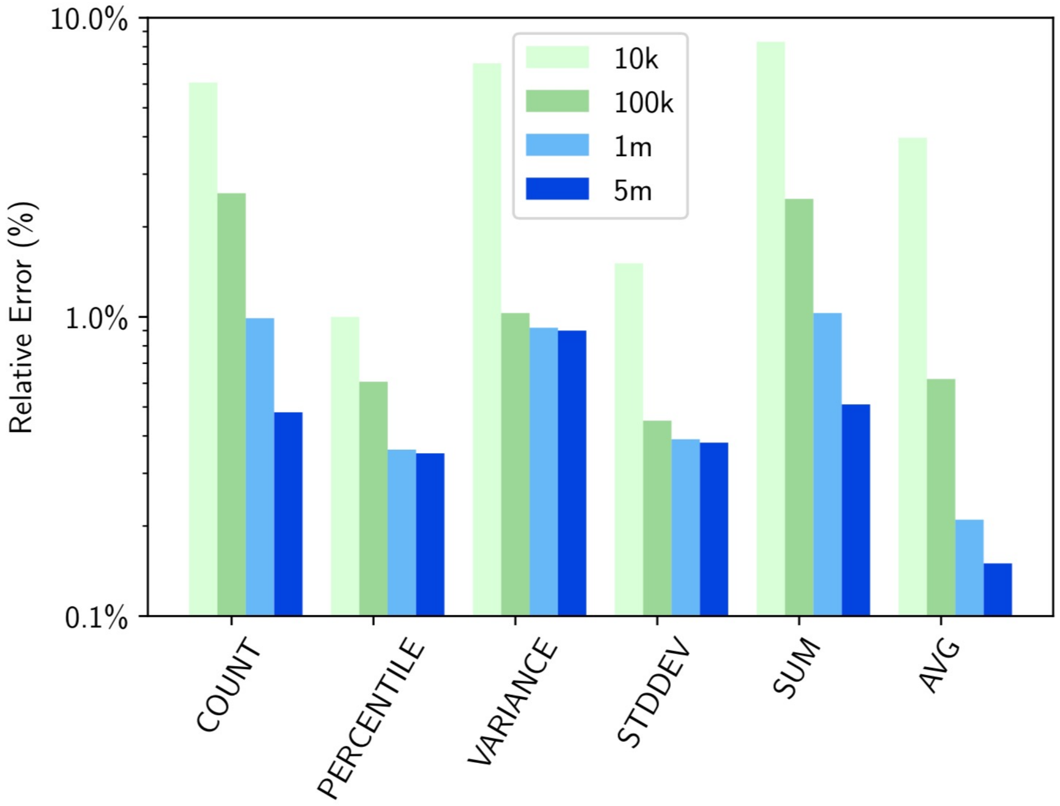
Dataset: TPC-DS
Sample size: 100k
540 synthetic queries
Column pair:
[ss_list_price, ss_wholesale_cost]



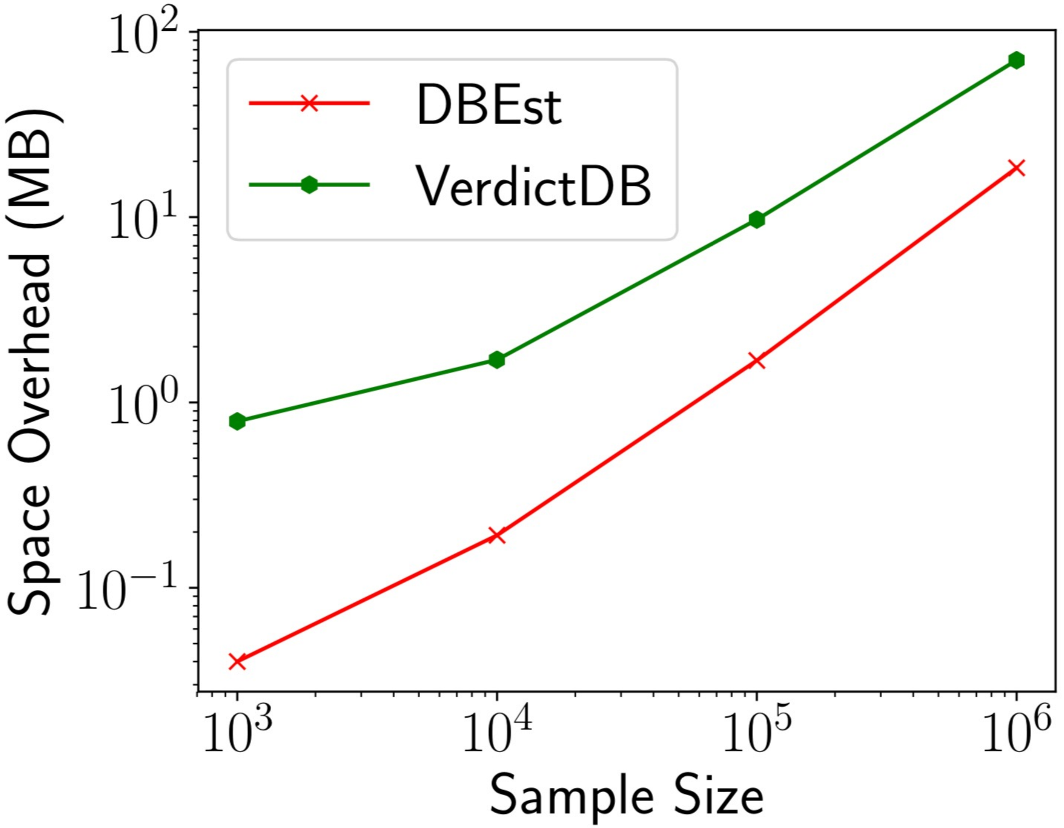
Influence of query range on relative error

Performance – Sensitivity Analysis
Sample size effect

Dataset: TPC-DS
Query range: 1%
1200 synthetic queries
Column pair:
[ss_list_price, ss_wholesale_cost]



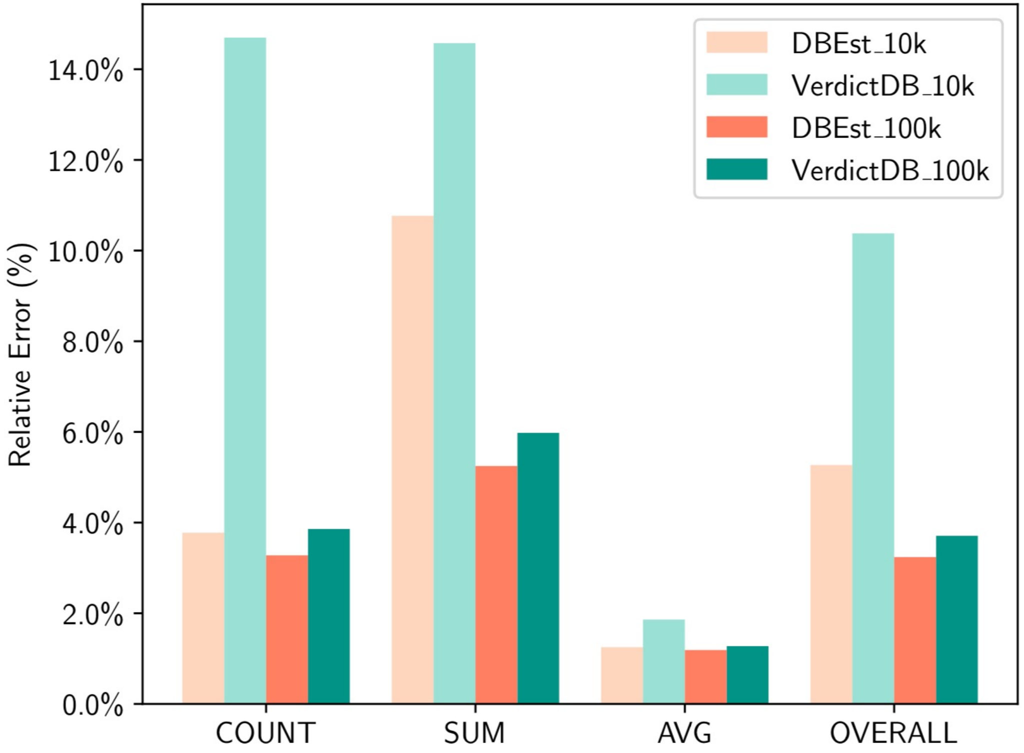
Influence of sample size on relative error



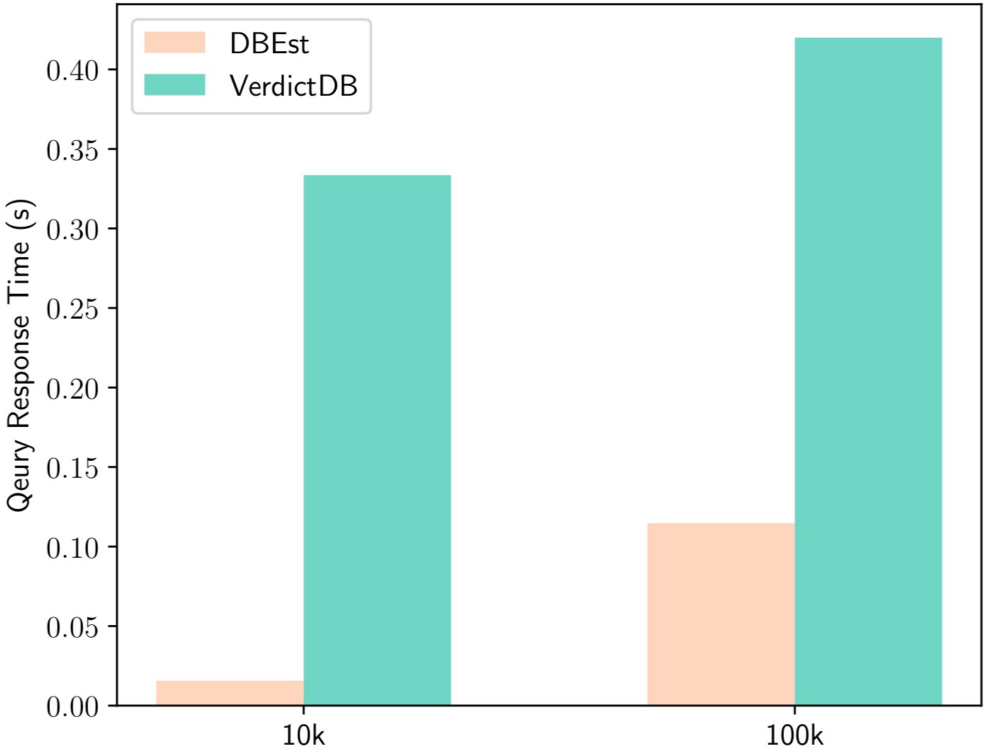
Influence of sample size on space overhead

Performance Comparison
TPC-DS dataset

Query range: 0.1%, 1%, 10%
~100 queries, involving 16 column pairs.
Sample size: 10k, 100k



Relative Error: DBEst vs VerdictDB

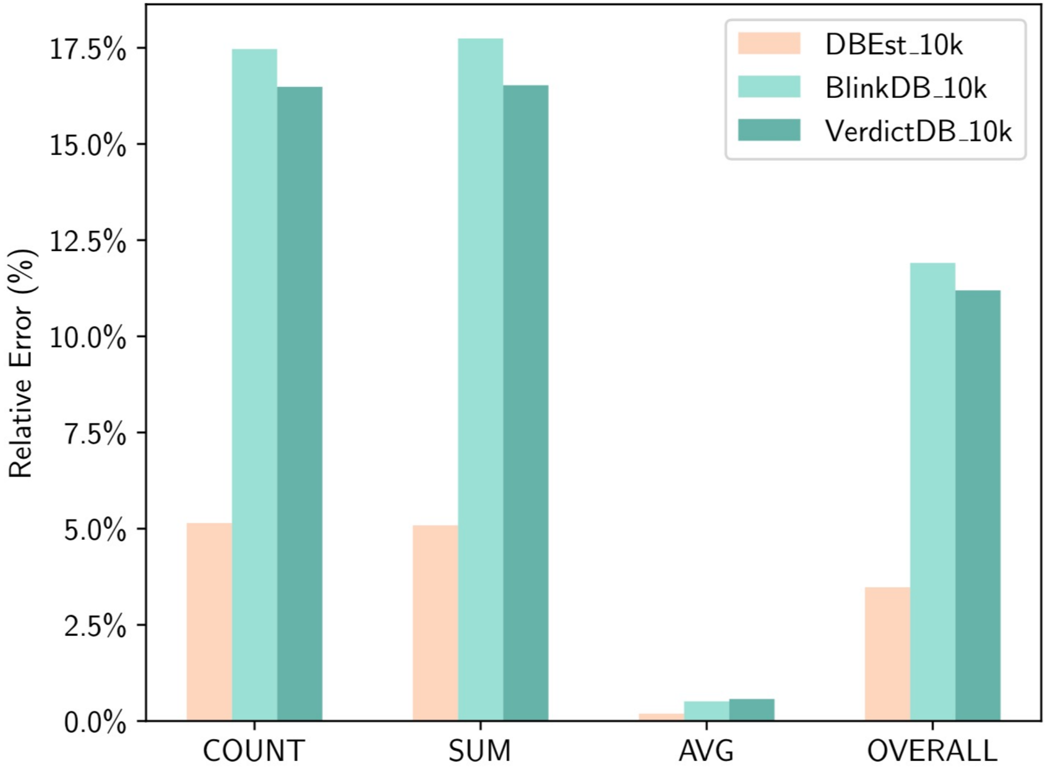


Query Response Time: DBEst vs VerdictDB

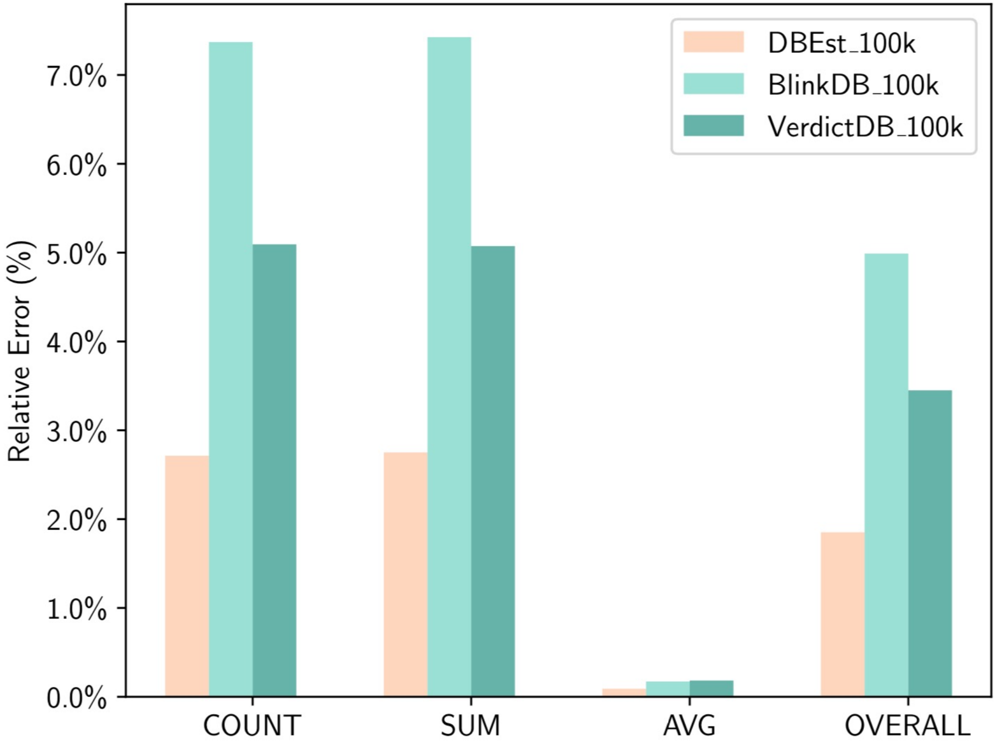
BOSTON UNIVERSITY

Performance Comparison
CCPP dataset

2.6 billion records, 1.4TB
Query range: 0.1%, 0.5%, 1.0%
108 queries, involving 3 column pairs.
Sample size: 10k, 100k



Relative error (10k sample)



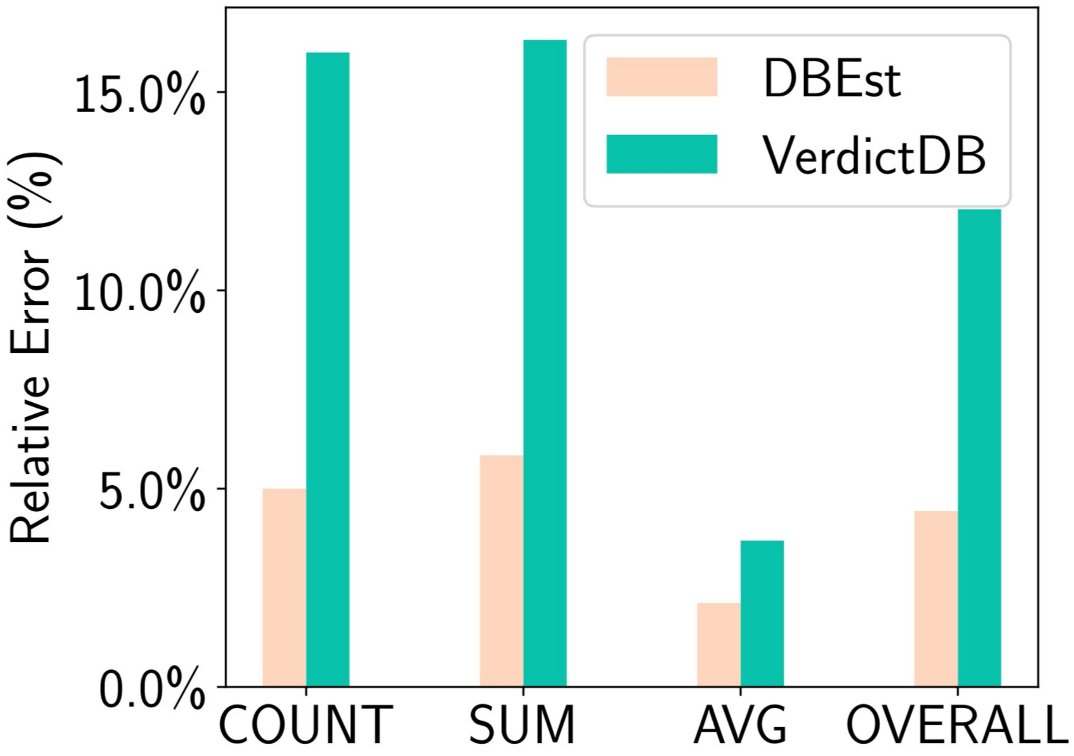
Relative error (100k sample)

BOSTON UNIVERSITY

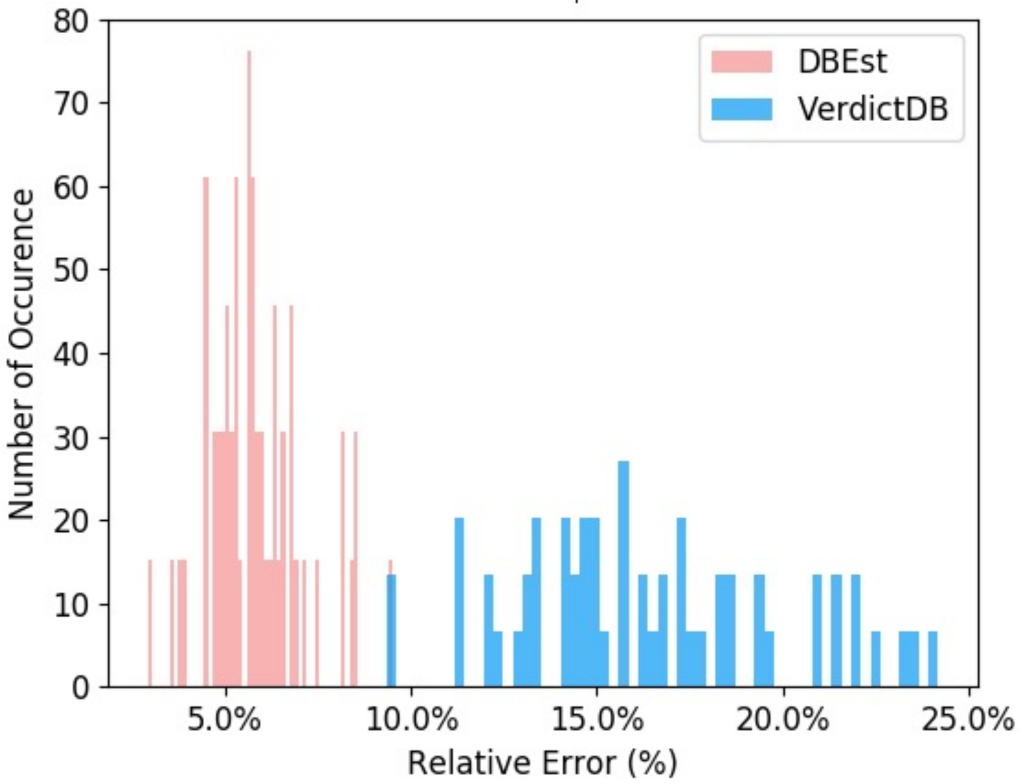
Performance Comparison Group By

SELECT AF(ss_list_price)
FROM store_sales
WHERE ss_wholesale_cost_sk …
GROUP BY ss_store_sk

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



Relative error for group by queries

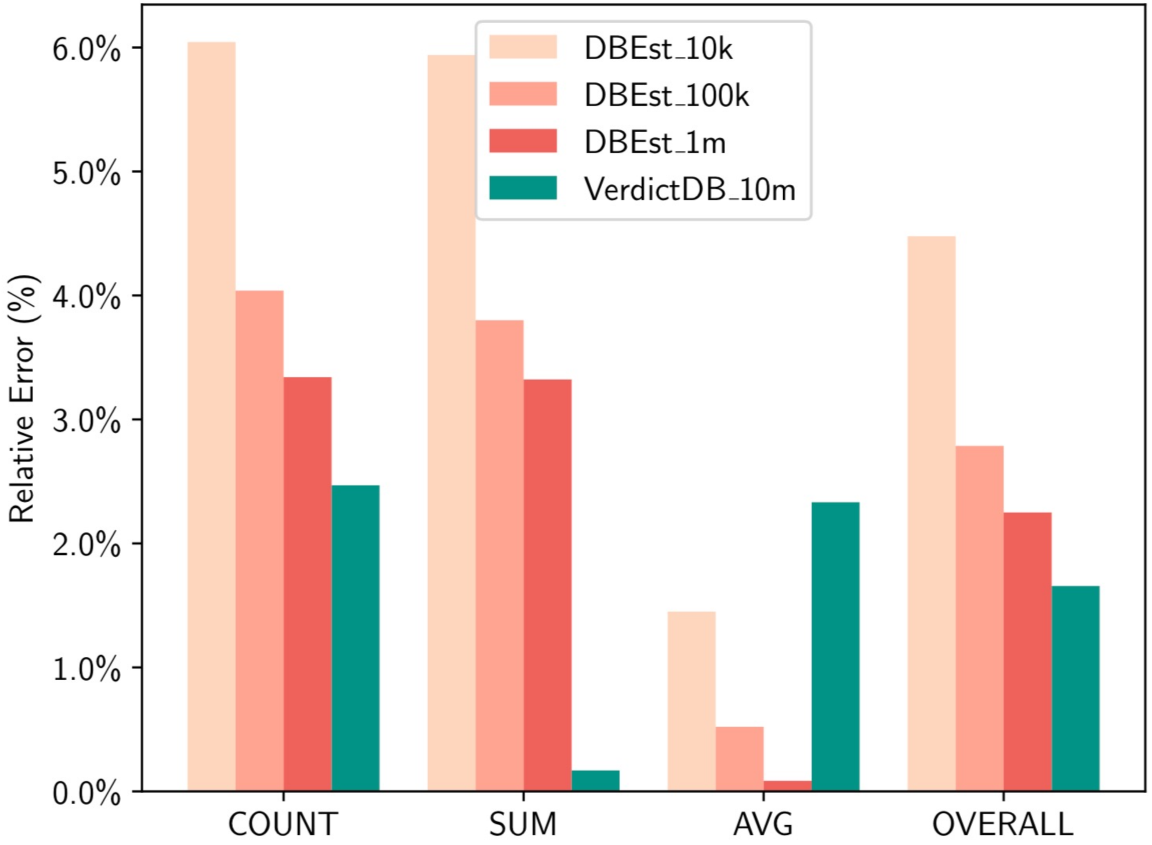


Accuracy histogram for SUM

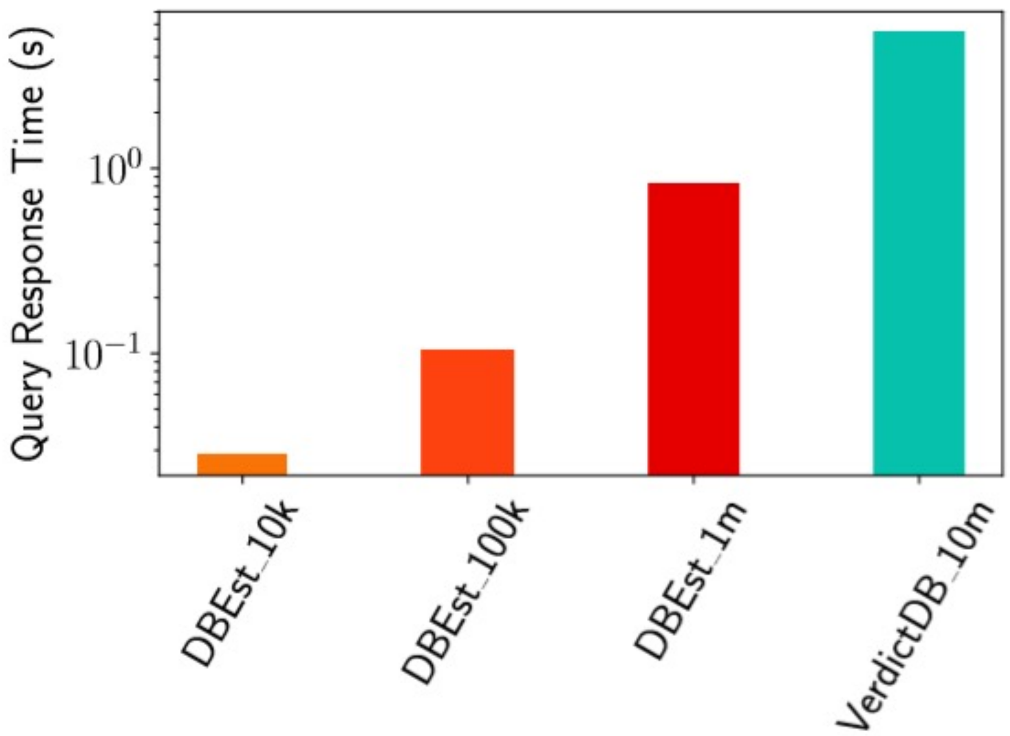
Performance Comparison Join

```
SELECT AF(ss_wholesale_cost), AF(ss_net_profit)
FROM store_sales, store
WHERE ss_store_sk=s_store_sk
AND s_number_of_employees BETWEEN …
```

- 42 queries.



Join accuracy comparison for the TPC-DS dataset

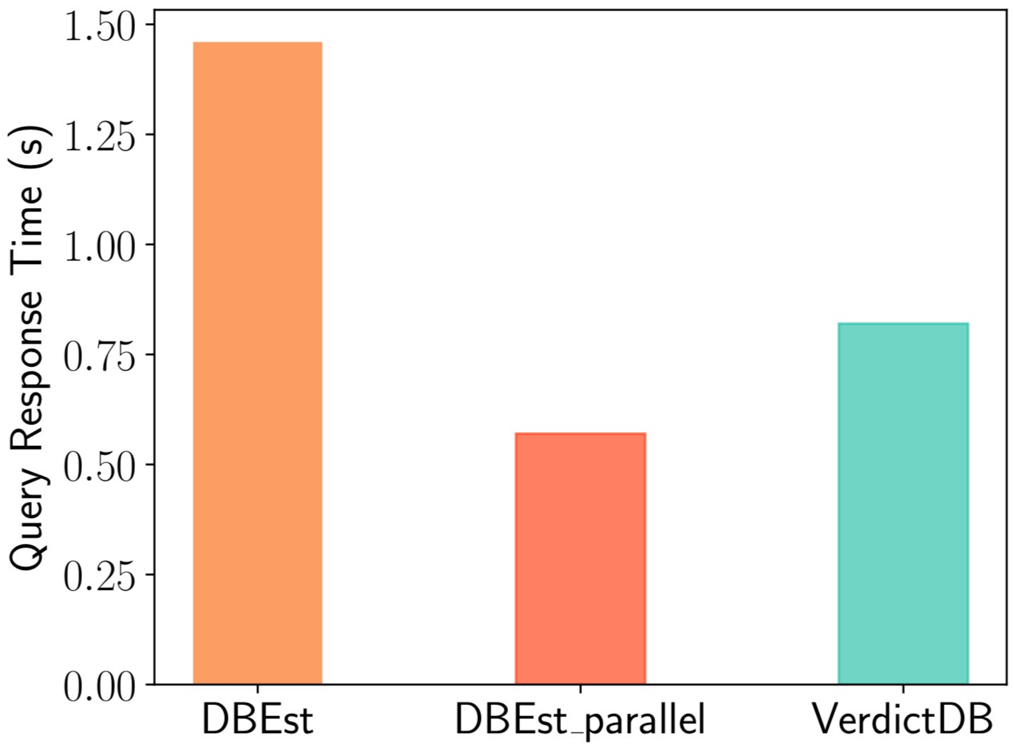


Query response time (s) for the TPC-DS dataset

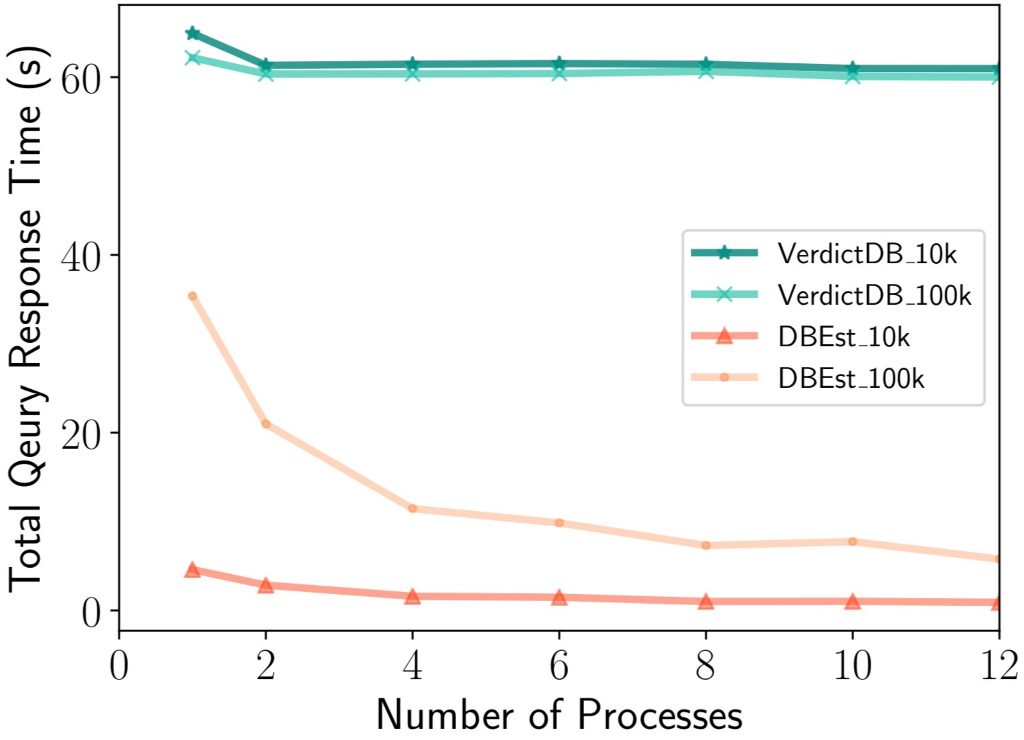
BOSTON UNIVERSITY

Parallel Query Execution

1 core versus 12 cores



Group by query response time reduction (TPC-DS)



Throughput of parallel execution (CCPP)

BOSTON UNIVERSITY

Limitations

- Group By Support ->too many groups
  - Model Training time ↑, Query Response time ↑, space overhead ↑.
- No error guarantee

Contribution & Conclusion

- Presented DBEst: a model-based AQP engine, using simple SML models:
  - Much smaller query response times
  - High(er) accuracy
  - Much smaller space-time overheads
  - Scalability
- Ensuring high accuracy, efficiency, scalability with low money investments -- resource (cpu, memory/storage/ network) usage.
- Future work: more efficient support for
  - Joins
  - Categorical attributes
  - Improved parallel/distributed DBEst

CS 561: Data Systems Architectures

class 24

Learned (Approximate) Query Processing

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

with slides from Marco Serafini and Peter Triantafillou