Model-Based Approximate Query Processing

Author: Futong Liu
Advisor: Sioulas Panagiotis, Sanca Viktor
Supervisor: Prof. Anastasia Ailamaki
Data-Intensive Applications and Systems Laboratory, EPFL

Abstract— **In the era of big data, databases increase tremendously in scale and it becomes excessively expensive to answer a query exactly. However, many analysis or applications require quick response from a query but still with high precision. Approximate Query Processing (AQP) is hence of great value as it can return query results within an error bound and a time limit. Traditional AQP methods use either online or offline sampling to make synopses of data, among which stratified sampling is arguably the sate-of-the-art. However, it still suffers from the problem of expensive sample construction, large space overhead, suboptimal accuracy and speed. This project explores the possibilities to build sample-less model-based AQP methods based on the statistics of the data, where three model-based approach are implemented with kernel density estimation (KDE), and compared with the sampling-based baseline. With a special focus on the weighted-KDE-based method, its properties, advantages and drawbacks are evaluated and analysed on the TPC-H benchmark.**

I. INTRODUCTION

With the proliferation of the IT industry, the scale of databases increases tremendously, and accordingly the full and exact execution of a query on the whole dataset takes arguably long time and significant hardware resources, which prohibits data analysts to carry out efficient exploration, interactive visualisation or reliable analysis of the dataset. However, in many applications, an approximate answer to a query is usually already enough to perform the job. For instance, interactive visualisation of a dataset needs to present the distribution of a data attribute within a specified range in short visual delays[3]. IoT datasets (eg. smart home, smart city[4]) also provide huge data volume with measurements from sensors based on time, location and etc and hence require fast and accurate query implementation in order to explore the subspace of the dataset.

This necessitates a framework that can timely and approximately answer a query within a fixed time bound and error bound, namely Approximate Query Processing (AQP). And fortunately, most of such analytical queries that are to be executed on the dataset are not ad-hoc and some prior knowledge about the query attributes can be concluded. For example, a very common template query in AQP has following structure:

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

where the query focus on a subspace of dataset T filtered by the predicate and AGG represents an aggregate function, such as COUNT, SUM, AVG and etc. Column y should be of numerical type and column x can be of numerical or ordinal (date, time and etc.) type.

Essentially, AQP works by compressing the dataset while preserving as much relevant information as possible with some knowledge about the query templates. Sampling has been the main stream of AQP and it mainly consists of two categories: online sampling and offline sampling. Online sampling[2] creates a sample of the dataset at the run time of a query and hence supports ad-hoc queries, but it can be expensive regarding hardware resource and inaccurate or even absent if the query is performed over rare sub-populations. Offline sampling[1] exploits the prior knowledge of columns that are to be queried upon and hence create samples based this information, which can mitigate the problem of rare sub-populations with biased sampling (eg. stratified sampling).

However, even a state-of-the-art sampling-based AQP is not perfect. Is it possible to build a new AQP approach that uses even less space than sampling-based one but still yields the same or even more accurate output? In the subsequent sections, the report will firstly introduce the rationale of Kernel Density Estimation (KDE) and briefly introduce the foundations of three possible model-based AQP models that utilise KDEs to make synopses of data. By testing these models on a TPC-H benchmark query, the performance (accuracy, response time, construction/training time, space and etc.) of the models will be compared. Lastly, the optimal KDE-based model will be evaluated and discussed in more details to get an overview of its performance over different queries and its properties and drawbacks under different circumstances.

II. OVERVIEW

A. *Density Estimation*

Essentially, AQP works by compressing the dataset while preserving as much relevant information as possible with some knowledge about the query templates. Model-based AQP proves to be efficient as it encapsulates the information of the dataset into a bunch of concise statistical models and uses these models to answer the query without touching the actual data nor its sample. Since well-trained statistical models are usually smaller and more compact than samples, it can reduce the foot prints to the memory and hence gain both time and space benefits.

When performing a simple AQP where the result attribute and the predicate attribute are of the same column, most aggregate functions (eg. expectation, variance and etc.) can be approximated once the probability density function of that column is available. Therefore, it is crucial to capture the probability density function as the model to describe the underlying distribution. There are mathematical distributions (eg. linear, normal, binomial and etc.) that uses parameters to describe the density function. However, in reality no hypothesis about the distribution of the data can be made, nor does the data even conform to a well-defined mathematical parametric distribution.

On the other hand, non-parametric estimation does not make any assumption about the distribution of the data. For instance, a histogram is widely used for density estimation. It partitions the range of the data into a fixed number of bins and then count the occurrences of the data in each bin. However, a histogram is not smooth and it suffers from the problem that the density function it describes depends on the size of bins significantly. Furthermore, it is difficult for a histogram to describe a multi-dimensional density function.

Kernel Density Estimation (KDE) is another non-parametric estimation approach with the same rationale as a histogram. At each available data point, it uses this data value and the kernel bandwidth as the parameters for the kernel function. The finally-computed density estimation is the superposition of all kernel functions at each data point and it is normalised such that the integral of the density function is one. Both the type and bandwidth of the kernel function can influence the quality of the estimation.

B. *Queries on One Single Column*

Consider a simple query where only one column is involved, as shown below:

```
SELECT agg_func(x)
FROM Table
WHERE x BETWEEN lb AND ub
```

It is trivial to compute the COUNT aggregation on this interval by calculating the fraction this interval takes as long as N (the total size of the table) is memorised:

$$COUNT(x) = N \times \int_{lb}^{ub} f(x)dx$$

It also feasible to compute the expectation(average) on this specific interval:

$$AVG(x) = \frac{\int_{lb}^{ub} xf(x)dx}{\int_{lb}^{ub} f(x)dx}$$

The SUM of the attribute on this interval is hence:

$$SUM(x) = COUNT(x) \times AVG(x)$$

Notation	Description
agg_func	The aggregate function. For instance, COUNT, SUM, AVG and etc.
x	The variable (column) in the predicate, namely the filter attribute.
y	The variable (column) in the result, namely the aggregate attribute.
T	The table to be queried.
lb	The lower bound of x of the interval specified in the query.
ub	The upper bound of x of the interval specified in the query.
N	The total size (number of records) of the entire table T.
$f(x)$	The density estimator of column x, which is an approximation of its probability density function.
$f(x, y)$	The density estimator of columns x and y, which is an approximation of the join probability density function of the two variable x and y.
$R(x)$	The regression model trained on data pairs (x, y). It returns a value for y every time a value of x is given.
$f_y(x)$	The density estimator of x that is weighted by y.
S	the total sum S of y over the whole table T.

TABLE I: Mathematical Notations

C. *Queries on Two Columns*

The query below abstracts a common query template where the the result column and the predicate column are different:

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

1) Two Dimensional KDE (2D-KDE): It is intuitive to directly use the KDE to approximate the joint distribution of the two columns x and y, where one dimension is used to represent the predicate column (x) and the other dimension to represent the result column (y).

To compute the COUNT aggregate of column is to calculate the marginal distribution of column x and then the fraction it takes:

$$COUNT(y) = N \times \int_{-\infty}^{\infty} \int_{lb}^{ub} f(x, y)dxdy$$

To compute the expectation of column y on an interval:

$$AVG(y) = \frac{\int_{-\infty}^{\infty} \int_{lb}^{ub} yf(x, y)dxdy}{\int_{-\infty}^{\infty} \int_{lb}^{ub} f(x, y)dxdy}$$

The SUM attribute is simply the product of the average and the count:

$$SUM(y) = COUNT(y) \times AVG(y)$$

*2) **Regression-Based KDE**:* According to [6], it is also feasible to calculate the expectation of column y when a one dimensional density estimation $f(x)$ of column x and a regression model $R(x)$ of data pairs (x, y) are available.

The COUNT attribute can be computed from the density estimation of x since the count of x and y are the same:

$$COUNT(y) = COUNT(x) = N \times \int_{lb}^{ub} f(x)dx$$

A well-trained $R(x)$ can be considered as a continuous function of x such that at every point of x it can predict or recover the value of y. Hence, to calculate the expectation of $R(x)$ on an interval of x is:

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

Similarly, the SUM attribute is the product of the AVG and the COUNT.

*3) **Weighted KDE**:* A weighted KDE $f_y(x)$ is a density estimator of x that is weighted by y. If a traditional density estimator $f(x)$ is considered as a regression model such that it gives the relative occurrences of x at each point of x, then a weighted density estimator can be considered as a regression model such that at every point of x it gives the relative sum of y. For instance, let x represents the date and y represents the number of orders on each day, then on a specific date x_0, $f_y(x_0)$ represents the relative number of orders on that day x_0, while a traditional KDE $f(x_0)$ represents the relative count of the date x_0.

It is called a relative sum because this regression model is normalised such that the integration from minus infinity to infinity is one. Because of this property of unit integration, it is also appropriate to call this regression model a weighted KDE. To calculate the SUM attribute of y, it requires to calculate the fraction this interval (lb, ub) takes up when the total sum S of y over the whole table is memorised:

$$SUM(y) = S \times \int_{lb}^{ub} f_y(x)dx$$

To calculate the COUNT attribute, it requires a KDE $f_x(x)$ that is weighted by the count of itself, which is essentially a traditional un-weighted KDE of x.

$$COUNT(y) = COUNT(x) = N \times \int_{lb}^{ub} f_x(x)dx$$

The AVG attribute is then simply the division:

$$AVG(y) = \frac{SUM(y)}{COUNT(y)}$$

*D. **Queries on multiple columns***

When there are more than one column in the result, it requires to build a bank of models where each one corresponds to a column. The result of the query is hence computed by calling each model in the model bank and combining the results into a data frame.

On the other hand, if there are more than one column in the filter(the "Where" clause), the model needs to be modified by increasing its dimension to represent these columns. For instance, a query with two columns in the filter and one column in the result:

```
SELECT AVG(z) FROM T
WHERE x BETWEEN lbx AND ubx
AND y BETWEEN lby AND uby
```

The 2D-KDE model should be augmented to 3D to describe the 3D joint distribution and hence:

$$AVG(y) = \frac{\int_{-\infty}^{\infty} \int_{lby}^{uby} \int_{lbx}^{ubx} zf(x, y, z)dxdydz}{\int_{-\infty}^{\infty} \int_{lby}^{uby} \int_{lbx}^{ubx} f(x, y, z)dxdydz}$$

The regression-based method needs to augment both the KDE and the regression model to two dimensional and hence:

$$AVG(y) = \frac{\int_{lby}^{uby} \int_{lbx}^{ubx} R(x, y)f(x, y)dxdy}{\int_{lby}^{uby} \int_{lbx}^{ubx} f(x, y)dxdy}$$

And the weighted KDE should update the KDE to two dimensional as well such that it can compute the SUM, the COUNT and then the AVG:

$$SUM(y) = S \times \int_{lby}^{uby} \int_{lbx}^{ubx} f_z(x, y)dxdy$$

*E. **Supporting "Group By" clause***

All the three KDE-based methods motioned above can support a "Group By" clause by treating each group specified by this "Group By" clause as a dataset itself. Namely, it only requires to build separate models on each group during training. For result approximation, it calls the pre-trained models in each group and computes the result. The final result data frame is the concatenation of the results in each group.

III. **IMPLEMENTATION**

This section explains the Python implementation details of all these methods.

*A. **Stratified Sampling***

The implementation of stratified sampling is based on the heuristic of BlinkDB[1]. A Query Column Sets (QCS) describes the set of columns that appears in the "Where", "Group By", "Having" clauses and it partitions the dataset into different strata. Stratified sampling over-represents the rare stratum and under-represents large stratum. With the user-defined error and confidence interval, stratified sampling can approximate the number of records to be sampled in each stratum according to [5]. Finally, a scalable sampling method [7] is applied in each stratum to generate the stratified sample.

Metric	Exact Result	Stratified Sampling*	2-D KDE	Regression-based	Weighted KDE
Construction Time	0s	121.61s	800s	11.20s	5.37s
Execution Time	5.62s	3.32s	200min	21.30s	2.51s
Space	1458.805MB	2.876MB	600MB	0.556MB	0.364MB
Relative Error	0%	1.57%	0.32%	1.89%	0.06%
Standard Deviation of Error	0%	0.67%	0.19%	1.38%	0.05%

TABLE II: Performance Comparison on Query 1

B. *Kernel Density Estimation*

A KDE uses a tree structure for fast data searching. Scientific packages in Python such as Scikit-Learn and Scipy both provide KDEs. Both of them supports multi-variate data. Scikit-Learn also supports multiple types of kernels (Gaussian, tophat, epanechnikov and etc.) while Scipy only supports a Gaussian kernel. However, the bandwidth of the kernel plays an even more significant role than the kernel type itself because the actual KDE is the summation of kernels. The estimation and evaluation of the bandwidth can be done by a rule of thumb and Scipy provides two automatic methods Scott[8] and Silverman[9] that can choose the bandwidth for the kernel by rule of thumb. Additionally, the bandwidth of the KDE in Scipy can be reset to another value even after the KDE model is trained. In the project, Scipy's KDE model is used.

C. *Regression Model*

The regression model used in the regression-based method is of great importance to the accuracy of the result. However, mostly the relationship between the independent variable x and independent variable y is noise-like, as there is no practical meaning nor relationship between the two, and even high-order polynomial regression model does not fit the data well. The selection of regression model is not a trivial task. Different models work better for different data regions, as mentioned in [6], where an ensemble method and a classifier are used for model selection. A KNN regression model uses the data itself for regression and hence fits the data quite well by sacrificing space. In this project, it is used for test purpose.

IV. EVALUATION

This section firstly compares all the methods (stratified sampling, 2D-KDE, regression-based method, weighted KDE) along with the exact query result. Then, it evaluates the performance weighted KDE method in more details and discusses its sensitivity against other factors, such as selectivity of query, kernel bandwidth and etc.

A. *Experiment Setup*

TPC-H is a benchmark to evaluate the performance of a database system. It contains business-oriented queries and data generated for commercial purposes. Each of the 22 queries in the TPC-H benchmark is a query template where the query structure is fixed and several parameters can be changed. In the project, since all the methods are only tested locally, so a scale factor of 1 (corresponding to approximately 1GB data) is selected. Experiments are run with PySpark locally on a MacBook with a 2.3 GHz Quad-Core processor, 16 GB memory and 256GB disk.

*Estimated from a subset of data.

B. *Performance Comparison*

Query 1 gives a brief report of prices in a fixed interval of ship dates grouped by return flag and line status.

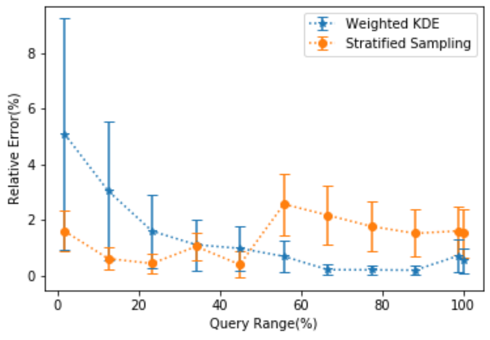
Table II shows the experiment results of each method executing query 1. Construction time is the sample-building time for stratified sampling and model training time for the other methods. Space is the sample size and the model size respectively. The relative error is computed by taking the mean of the relative error on all result attributes, among which the standard deviation is computed as well.

All the methods successfully extracted information from the original data. However, 2D-KDE takes a very long execution time, where the bottleneck is the double integration. Although with a one dimensional integration, regression-based method also takes quite some time as it needs to inquire the regression model at each data point during the integration; the quality of the regression model also needs to be fine-tuned for a satisfactory relative error. The weighted KDE is most apt method here as it is simple and fast to build, fast to execute and generates high precision results. Hence, the sections below will mainly discuss the performance of the weighted KDE method and compare it with stratified sampling.

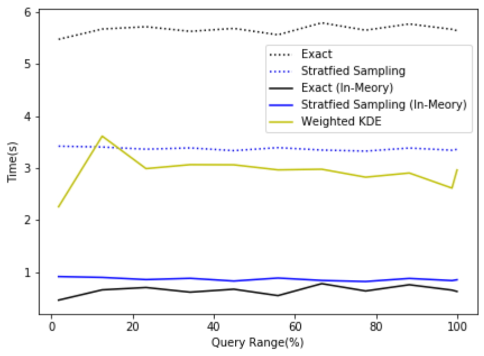
C. *Query 1: Effect of Selectivity and Kernel Bandwidth*

Figure 1 presents the effect of selectivity of the predicate and the kernel bandwidth. The performance of weighted KDE degrades when only a small part of the dataset is included by the filter, as shown in figure 1a. This can be further elaborated in figure 2: when a large portion of data is queried the kernel bandwidth does not influence the accuracy of the result significantly, while the kernel bandwidth influences queries over small portion of data significantly. Since there is over-fitting not a problem for AQP, the kernel bandwidth should be selected small when possible.

Figure 1b shows that the selectivity of predicate does not influence the time performance notably. The dotted line shows the execution time of the exact query and sample-based method including the data transfer time from the disk to the memory. In the project, everything was run locally and both the stratified samples and the KDE models can fit in the memory. But in reality, the samples might be so large that disk access might be required for answering the query.



(a) Relative Error and Selectivity



(b) Execution Time and Selectivity

Fig. 1: Effect of Selectivity

Therefore, the space advantage of the KDE-based method might win itself even more speed advantage.

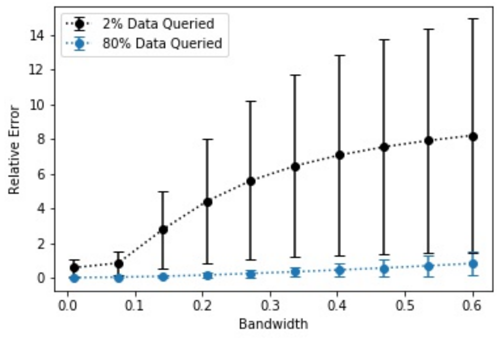
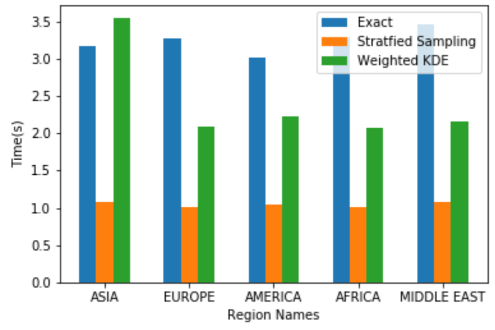


Fig. 2: Effect of Kernel Bandwidth

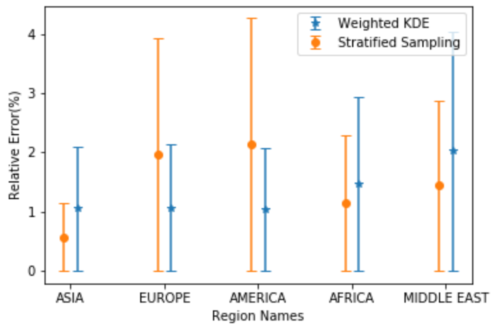
D. *Query 5: Richness of models*

Query 5 reports the revenue of one year in each country in a user-given region. There is a nominal column (region) in this query. To support predicates over nominal columns, to model takes this column as a ”Group By” column and stores this information in the catalogue that keeps track of all the KDE models that have been created.

Figure 3 shows the execution time and the relative error in each region. Both stratified sampling and weighted KDE



(a) Time and Region



(b) Relative Error and Region

Fig. 3: Query 5:Richness of models

model can improve the execution speed of the query. Figure 3b shows that the relative error of regions ”AFRICA” and ”MIDDLE EAST” are higher than the other.

What factors influence the performance of the weighted KDE model? By querying the whole dataset it is known that there are exactly five countries in each of the five regions. Additionally, the number of records in each region are almost the same. The range of order dates are also the same across these regions.

Region Name	STDDEV of Countries's Count
AMERICA	228.194435
EUROPE	251.674393
ASIA	476.581263
MIDDLE EAST	578.440835
AFRICA	677.098442

TABLE III: Models' degree of dispersion in each region

Table III shows the standard deviation of the number of records in each country. Since there are five KDE models created for each region, this standard deviation can be used to evaluate the degree of dispersion of the models in each country. A high value of standard deviation means that among the five models, some models are rich (trained from large amount of records) and some are poor ((trained from small amount of records). The poor models hence becomes the bottleneck of the performance, for example both ”AFRICA” and ”MIDDLE EAST” has a large value of standard deviation.

E. *Query 6 †: Curse Of Dimensionality*

Query 6 reports the revenue in a specific data interval, a specific discount range and below a specific quantity. However, this is not a trivial task for KDE models to be trained on as it includes four dimensions: three dimensions for predicates and one dimension for the final result.

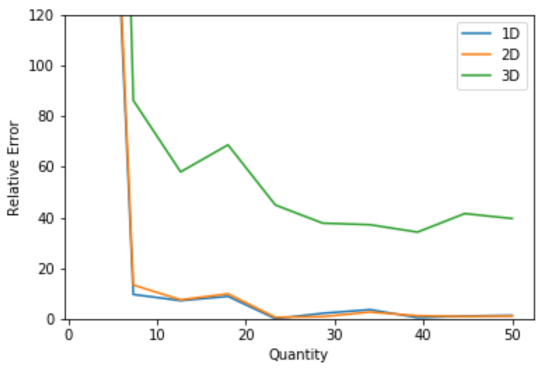


Fig. 4: Effect of Dimensionality

Figure 4 shows how the relative error changes with respect to the quantity predicate under different dimensions. The 1D case uses only the quantity column for filtering and integrates on the KDE with the other two dimensions fixed (from minus infinity to infinity). The 2D case fixes the date predicate as from minus infinity to infinity and hence uses two dimensions (date and quantity) for filtering. The 3D case does not fix any dimension. It can be seen that the performance of the KDE severely degrades when the all dimensions are used for filtering.

The kernel function leaks more information with the increasing of dimensions. If the 1D case is considered as integrating over an interval and the 2D case as over a surface, then the 3D case is to integrate over a volume, where he kernel can leak information over all three dimensions. To simply make the kernel bandwidth smaller does not help with the problem as it increases the difficulty and the error of the integral.

F. *Query 10 ‡: Effect of "Group By"*

Query 10 returns the customers who have returned an order. Since all the information of the customer are used in the "Group By" clause, this results in a very large amount of groups in the result. The model-based method creates a model for each group and saves the group information into a catalogue. The final result is the concatenation of the result of each group. Figure 5 shows how the number of groups influence the construction time and the execution time. With the increasing of the groups, the construction time increases accordingly to train the models.

G. *Limitations*

During the implementation and testing of the weighted-KDE-based method, there are several drawback identified:

‡This query is not executed on the full dataset.

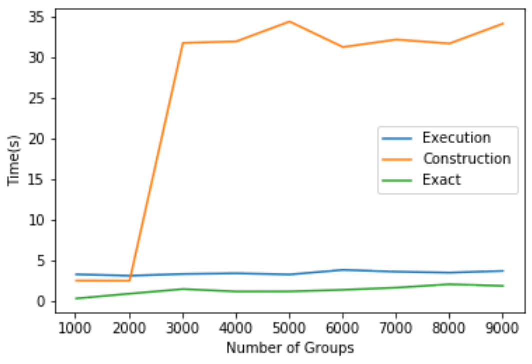


Fig. 5: Effect of the Number of Groups

- The KDE-based method assumes that the attributes in the SELECT clause are predictable such that the corresponding KDE model can be pre-trained. In other words, it does not support user-defined functions, in contrast to sampling-based method.
- The KDE-based method does not distinguish $<$ and $<=$ and other corner cases.
- The KDE-based method can only support attributes of numerical or ordinal type for results (the "Select" clause).
- If there are updates to the dataset, for example to add new records, The KDE-based method need to be retrained to preserve the model constraints.
- The KDE-based method does not support the pre-definition of estimation error and confidence interval.

V. CONCLUSIONS

In this project, there are three local prototypes of model-based AQP methods that are implemented and compared, namely the 2D-KDE, regression-based KDE and weighted KDE. TPC-H benchmark is used to compare the methods along with stratified sampling. Weighted KDE arguably shows the best performance as it is fast to construct, uses small memory space, responses swiftly and yields accurate result. It can outperform the stratified sampling method in some cases, but there are also cases where the KDE-based method encounter difficulties. Stratified sampling may after all be a suitable method for some AQP tasks.

Future work includes extrapolating the methodology to large distributed systems and testing the scalability of the methods. It can be envisaged that large database may bring the KDE-based method more advantages, as models are more compact than samples and hence does not require as many memory resources as sampling-based approach during construction and execution, which may win itself both space and time benefits. Additionally, the data can be skewed in a distributed system and the effect skewness should be further studied.

BIBLIOGRAPHY

[1] Sameer Agarwal et al. *BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data*. 2012. arXiv: 1203.5485 [cs.DB].

[2] Chris Jermaine et al. "Scalable Approximate Query Processing with the DBO Engine". In: *ACM Trans. Database Syst.* 33.4 (Dec. 2008). ISSN: 0362-5915. DOI: 10.1145/1412331.1412335. URL: https://doi.org/10.1145/1412331.1412335.

[3] Moritz Kulessa et al. *Model-based Approximate Query Processing*. 2018. arXiv: 1811.06224 [cs.DB].

[4] Xuan Liang et al. "Assessing Beijing's PM 2.5 pollution: severity, weather impact, APEC and winter heating". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science* 471 (Oct. 2015), p. 20150257. DOI: 10.1098/rspa.2015.0257.

[5] S.L. Lohr. *Sampling: Design and Analysis*. Advanced (Cengage Learning). Cengage Learning, 2009. ISBN: 9780495105275. URL: https://books.google.com/books?id=aSXKXbyNlMQC.

[6] Qingzhi Ma and Peter Triantafillou. "DBEst: Revisiting Approximate Query Processing Engines with Machine Learning Models". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 1553–1570. ISBN: 9781450356435. DOI: 10.1145/3299869.3324958. URL: https://doi.org/10.1145/3299869.3324958.

[7] Xiangrui Meng. "Scalable Simple Random Sampling and Stratified Sampling". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–531–III–539.

[8] David Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Mar. 2015. ISBN: 9781118575536.

[9] Bernard Silverman. *Density Estimation For Statistics And Data Analysis*. Vol. Vol. 26. Jan. 1986. DOI: 10.1007/978-1-4899-3324-9.