APPROXIMATE QUERY PROCESSING TECHNIQUE FOR EXECUTING JOIN-AGGREGATE QUERIES ON BIG DATA

Praveen Kumar Sadineni
Mahantech Corporation, Charleston, WV, USA
praveenkumarsadineni1998@gmail.com

Abstract - Big Data query processing mainly deals with executing queries on Big Data. Since, most of the Big Data repositories are made up of unstructured data, coupled with high velocity and volume of data generation, designing efficient query processing techniques imposes significant challenges. Hence, Approximate Query Processing Techniques (AQPTs) are an attractive option. AQPT are ideally suited for executing aggregate queries, where the AQPT provides approximate results with attractive computational efficiency. Recently in the literature, AQPT was presented to execute simple non-join aggregate queries on Big Data. However, this presented AQPT does not deal with the more complex join-aggregate queries. Hence, in this paper, AQPT is presented for the approximate execution of join-aggregate queries. The proposed AQPT is designed using Central Limit Theorem (CLT), and achieves predefined estimation error. An empirical analysis study is presented in which the proposed AQPT is compared against a contemporary technique. In this empirical analysis study, the proposed AQPT significantly outperforms the contemporary technique both in-terms of estimation accuracy and computational latency.

Keywords: Approximate Query Processing, Big Data, Sampling, Aggregate Join Queries.

1. Introduction

1.1. *Overview*

Big Data refers to that complex data which is characterized by substantial Volume, and which is generated with extreme Velocity. Currently, Big Data has become quite common, and the main contributors of Big Data are: Social Networks, Wireless Networks, Remote Sensing Devices etc. In most of the cases, Big Data is generated in unstructured format [1]. Due to the unstructured format of Big Data, it becomes difficult to build efficient query processing techniques. The primitive choice is to transform the unstructured data into structured data, and store this transformed data in Relational Database Management Systems (RDBMS), and thus, efficient query processing can be achieved by utilizing the power of Declarative Language (SQL) and efficient data processing algorithms built inside RDBMS. However, this primitive choice is largely impractical for Big Data because, the data is generated at extreme volume and velocity, and due to which, infeasible amount of computational resources are required.

The Map-Reduce framework [1] is specifically utilized to achieve efficient query processing on Big Data. The Map-Reduce exploits parallelism in achieving high query-execution efficiency. Specifically, there are two types of tasks involved in query execution: Map and Reduce tasks. The Map tasks, generally are involved in operations such as: data-tuple access, sorting, filtering etc., whereas the Reduce tasks are involved in operations such as: aggregation, counting etc. Each Map task can be executed in a separate computing node, and maximum efficiency can be achieved if each Map task is independent of the other. Similarly, the Reduce tasks can also be executed concurrently with each other. It must be noted that, many Big Data applications which execute using Map-Reduce, utilize data size in order of Tera Bytes or Peta Bytes, hence, to achieve substantial query execution efficiency, infeasible computational resources are required [1]. Thus, novel solutions have been proposed in the literature to achieve better query-execution efficiency such as: approximate query processing [1,2], which provide approximate answers to aggregate queries.

1.1. *Addressed Open Issues*

$$\textbf{SELECT}\ \text{aggregate}(R.A)$$
$$\textbf{FROM}\ R$$

Query 1: Simple Aggregate Query (SQL)

Approximate aggregate query processing techniques presented in [1,2] provide approximate results to a simple non-join aggregate query as depicted in Query 1, for Big Data queries. Here, *aggregate()* denotes the aggregate function such as: Sum, Average, Variance, Standard Deviation etc. Additionally, individual selection predicates can be imposed on relation *R*. Specifically, in [1], the presented approximate query processing technique utilized *Central Limit Theorem (CLT)* [1].

Let, Q denote a simple non-join aggregate query, $F(Q)$ denote the result of Q, $est(Q)$ denote the estimated result of Q, ε denote the estimation error and p denote the estimator confidence. Then, the approximate query processing technique presented in [1] achieves the estimation quality as represented in "Eq. (1)".

$$P(|F(Q) - est(Q)| \leq \varepsilon) = p \quad (1)$$

In [1], the presented Approximation Query Processing Technique (AQPT) addresses four aggregate operations: Sum, Average, Variance and Standard Deviation. This AQPT is designed using Map Reduce framework and operates on unstructured data. Also, AQPT requires generation of random samples. For the AQPT presented in [1], random samples are generated by simple table scan, because the addressed queries are Non-Join Aggregate (NJA) queries.

$$\textbf{SELECT}\ \text{aggregate}(R_i.A)$$
$$\textbf{FROM}\ R_1, R_2, \ldots . R_k$$
$$\textbf{WHERE}\ Join(R_1, R_2, \ldots R_k)$$

Query 2: Join Aggregate query (SQL)

Consider a join-aggregate query depicted in Query 2. Here, there are k relations denoted as $(R_1, R_2, \ldots R_k)$ and $join()$ denotes the join conditions imposed on $(R_1, R_2, \ldots R_k)$. Also, $R_i \in (R_1, R_2, \ldots R_k)$. Additionally, individual selection predicates can be defined over $(R_1, R_2, \ldots R_k)$.

The AQPT presented in [1] cannot be directly applied in executing Join Aggregate (JA) queries, and many significant challenges have to be resolved which are described below:

1. The random samples for JA queries cannot be generated by using the approach used for NJA queries, because join is a computationally complex and costly operation. Hence, a new random sample generation scheme has to be designed.
2. The random sample generation scheme has to be designed by using Map Reduce framework, and this scheme should achieve noticeable computational efficiency. Otherwise, the AQPT will become impractical.
3. The estimation quality for JA queries should satisfy Equation 1, and also achieve this goal by consuming limited computational cost. In [1], the presented AQPT achieved both these goals. However, since, join is a complex operation, substantial design improvements have to be performed on the AQPT presented in [1] in-order to achieve the dual goals of estimation quality (as represented in Equation 1) and computational efficiency.

Until now in the literature [1,2], AQPT for JA queries on Map-Reduce framework has not been presented, and this main open issue is addressed in this paper. The presented AQPT in this paper considers four query types based on the aggregate operations: Average, Sum, Variance and Standard Deviation, and these are denoted as: Query Type 1, 2, 3 and 4 respectively. The presented AQPT is designed using CLT and achieves estimation quality as represented in Equation 1. A random sample generation scheme is presented to generate random samples from the result set of JA queries, which will be used for result estimation. This random sample generation scheme is specifically designed for the Map Reduce framework and achieves noticeable computational efficiency. The merits of the presented AQPT in-terms of computational efficiency and estimation quality are established both theoretically and empirically.

1.2. *Paper Contributions*

The following contributions are presented in this paper:

1. AQPT for JA queries on Map-Reduce framework is presented, along with the random sample generation scheme. The query types 1, 2, 3 and 4 are addressed in the presented AQPT. Theoretical bounds on estimation quality are established and presented.
2. The empirical analysis study on the proposed AQPT is presented. This empirical analysis study is conducted on TPC-DS dataset [1,2]. In this empirical analysis study, the proposed AQPT is compared against a contemporary technique, and the relative merits of computational efficiency and estimation quality of the proposed AQPT is presented.

This paper is organized as follows: Section 2 describes the related work corresponding to the main open issue addressed in this paper; Section 3 presents the proposed AQPT; the empirical analysis study of the proposed AQPT is presented in Section 4; finally, the paper is concluded in Section 5.

2. Related Work

In [3], the initial work on AQPT for aggregate queries on RDBMS is presented. This work exclusively focuses on structured data and RDBMS. Theoretical quality bounds for estimated results are presented. However, this work does not address unstructured data or Big Data scenario.

The aggregate query based AQPT presented in [4] exclusively focused on a specialized RDBMS called DBO database system. Here, when an aggregate query is being executed, a guess of the final answer is being computed concurrently, so that, the user can stop the query execution, if the estimated answer is sufficient.

In [5], a tool for interactive query processing is presented. This tool is similar to the work presented in [4] where during the execution of aggregate queries, a guess of the final answer is provided to the user, and the quality of the guess is being improved continuously. The user can switch off the execution in-between, if the estimated answer meets the user requirements WRT estimation quality.

One of the fundamental questions in RDBMS query execution is to measure the progress of the query during its execution. In [6], a technique to measure the progress of query execution is presented. For the aggregate query, the user can view the query execution progress and estimated result. If the leftover execution latency is large, and the user cannot bear this latency, then, the user can switch-off the query execution and settle for the estimated result.

In [7], an almost identical aggregate query execution progress estimator technique is presented. The goals of this presented technique are exactly identical to the goals presented in [6], which is to provide the user with leftover execution latency, and providing the user to switch-off the query execution, if the quality of the estimated result is good enough.

Some aggregate queries can take extremely long time to be completely executed [8]. Hence, in [8] a technique to estimate the duration of the query execution is presented. This technique estimates the duration even before the query is executed. This estimated information aids the user to decide whether to opt for complete query execution or approximate query execution. This presented technique has specifically focused on aggregate queries.

In [9], learning based AQPT for aggregate queries is presented. This presented technique utilized the concept of Histograms [10]. Here, the execution latency of the given query is estimated through the aid of Histograms and learning algorithms. If the execution latency is prohibitively large, then, the user can opt for approximate execution of aggregate queries.

One of the earliest contributions presented in the literature to design an aggregate query based AQPT is presented in [11]. This presented technique utilizes the concept of adaptive sampling. Here, the result estimation is performed through sample sizes rather than number of samples. The main limitation of this presented technique is that, it requires indexes to achieve efficiency in estimation process. Also, this presented technique is exclusively focused on structured data and RDBMS.

The AQPT presented in [12] is an extension to the technique presented in [11]. This presented technique focuses on join-aggregate queries. However, this technique, as in [11], focuses on structured data and RDBMS.

The distinct value calculation queries are used for calculating the distinct values in the given table attribute. The technique presented in [13] aims at estimating the distinct values in the given attribute. However, this presented technique only focuses on structured data and RDBMS.

The aggregate query based AQPT presented in [14] focuses on only one aggregate operation: count operation. It can also be considered as an extension to the technique presented in [11]. The same limitation present in the technique presented in [11] is also present here, which is, the presented technique in [14] focuses solely on structured data.

The join-aggregate query based AQPT presented in [15] improves the result reported in [12] in-terms of estimation quality. Additionally, the presented technique in [15] consumes relatively lesser number of samples when compared to the technique presented in [12].

The distinct value estimation technique presented in [16] improves the result reported in [13] in-terms of estimation accuracy. As in [13], this presented technique in [16] only focuses on structured data and RDBMS.

The AQPT requires that, random samples need to be produced for estimating the result set of aggregate queries. Hence, in [17], a technique is presented to produce random samples in Map Reduce framework. However, in this presented technique, the theoretical bounds for the estimation quality has not been presented.

An alternate framework to execute aggregate queries is sample data query execution [18,19,20,21,22]. Here, the given aggregate query is not executed on complete dataset, however, it is executed on data samples. This framework has been presented in [18,19,20,21,22]. However, the work presented in [18,19,20,21,22] do not provide the theoretical bounds for the estimation quality.

From the above describe literature survey, it is clear that, even though AQPT for aggregate queries have been presented for RDBMS, along with theoretical bounds of estimation quality, these presented techniques cannot be directly utilized for Big Data scenarios, because of the unstructured data format in Big Data. There have been some contributions in the literature to design Map Reduce based AQPT for aggregate queries. However, these techniques do not provide theoretical bounds for the estimated result quality. Hence, in subsequent section of this paper, a Map Reduce based AQPT for join-aggregate queries will be presented, in which theoretical bounds of the estimation quality will also be presented.

3. Proposed AQPT for Join-Aggregate Queries

3.1. *Random Sample Generation Scheme*

The Query 2 will be considered as the reference query to describe the proposed AQPT. Let, $R_j \in (R_1, R_2, \dots R_k)$. The unstructured data for R_j will be stored in blocks. Let, the number of blocks which contain the data corresponding to R_j be m_j. The value of m_j will be large, and the reason for this choice will be outlined subsequently. The i^{th} block will be denoted as B_i^j. The number of tuples in B_i^j will be denoted as $|B_i^j|$. The y^{th}tuple of B_i^j will be denoted as $t_{i_y}^j$, and the entire tuple set will be denoted as $\{t_i^j\}_{y=1,2,\dots|B_i^j|}$. Consider the block set $(B_{i_1}^1, B_{i_2}^2, \dots . B_{i_k}^k)$. The joined tuple set corresponding to the block set according to the defined conditions in $Join(R_1, R_2, \dots R_k)$ will be denoted as $\{T_{i_1,i_2,\dots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)}$. Here, $NJ(i_1, i_2, \dots i_k)$ denotes the cardinality of the operation $Join(B_{i_1}^1, B_{i_2}^2, \dots . B_{i_k}^k)$.

Algorithm 3.1. $Random\_Sample\_Generator(Q)$

$\boldsymbol{for}(j = 1; j \leq k; j++)$

$B_{i_j}^j = random\_block\_select(B_1^j, B_2^j, \dots B_{m_j}^j)$

$assign\_block(Map(B_{i_j}^j))$

$\hat{B}_{i_j}^j = block\_refine(Map(B_{i_j}^j))$

End for

$T^S = join\_sample(Reduce(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k), Q)$

Return (T^S)

Algorithm 3.2. $block\_refine(Map(B_{i_j}^j))$

$\hat{B}_{i_j}^j = structure\_block(B_{i_j}^j)$

Return $(\hat{B}_{i_j}^j)$

Algorithm 3.3. $join\_sample(Reduce(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k), Q)$

$\{T_{i_1,i_2,\dots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)} = join\_blocks(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k, Q)$

$\boldsymbol{If}(\{T_{i_1,i_2,\dots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)} \neq NULL)$

$T^S = random\_tuple\_select(\{T_{i_1,i_2,\dots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)})$

Return (T^S)

Else

Return $(NULL)$

The Query 2 will be denoted as Q, and its result will be denoted as $F(Q)$. The result set tuples (the tuples which are used in calculating $F(Q)$) of Q will be denoted as $Result(Q)$. Algorithm 3.1. outlines the technique to generate a random sample for Q, and this algorithm is denoted as $Random\_Sample\_Generator(Q)$. Here, random blocks are selected uniformly from each $R_j \in (R_1, R_2, \dots R_k)$, and this selection is accomplished through $random\_block\_select()$. Each selected block denoted as $B_{i_j}^j$ is assigned to a unique map task denoted as $Map(B_i^j)$ through $assign\_block()$.

The map task execution algorithm is outlined in Algorithm 3.2. which is denoted as $block\_refine()$. Here, the input block $B_{i_j}^{j}$ is structured through $structure\_block()$, which involves creating tuples from the unstructured data.

In Algorithm 3.1., the random sample for Q is created through the reduce task denoted as $Reduce(\hat{B}_{i_1}^{1}, \hat{B}_{i_2}^{2}, \dots \hat{B}_{i_k}^{k})$.The reduce task execution algorithm is outlined in Algorithm 3.3. which is denoted as $join\_sample()$. Here all the input blocks $(\hat{B}_{i_1}^{1}, \hat{B}_{i_2}^{2}, \dots \hat{B}_{i_k}^{k})$ are joined based on the join condition of Q through $join\_blocks()$, in-order to create $\{T_{i_1,i_2,\dots i_k}^{y}\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)}$. Finally, a random tuple denoted as T^S is selected from $\{T_{i_1,i_2,\dots i_k}^{y}\}_{y=1,2,..,NJ(i_1,i_2,\dots i_k)}$ uniformly through $random\_tuple\_select()$.

It must be noted that, in Algorithm 3.1., the blocks are selected through uniform random sampling, and similarly, T^S is selected form these blocks through uniform random sampling, hence, the output of Algorithm 3.1. can be considered as an appropriate random sample form $Result(Q)$.

In the next section, the AQPT corresponding to Query Type 1, 2, 3 and 4 will be presented. This AQPT utilizes the CLT, which is described below.

Consider a random variable X. Let, $(X_1, X_2, \dots, X_n)$ denote the random samples from X. Let, $E[X] = \mu$ and $Var(X) = \sigma^2$. The sample mean of X is denoted as $\bar{X}_n$ and it is represented in "Eq. (2)".

$$\bar{X}_n = \frac{(X_1 + X_2 + \cdots + X_n)}{n} \tag{2}$$

The unbiased sample variance of X is denoted as S_n^2 and it is represented in "Eq. (3)".

$$S_n^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X}_n)^2}{n-1} \tag{3}$$

The CLT is represented in "Eq. (4)".

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| \le \varepsilon) = 2\varphi\left(\frac{\varepsilon\sqrt{n}}{\sigma}\right) - 1 \tag{4}$$

Another important result which will be used in this paper is the Law of Large Numbers [1] which is represented in "Eq. (5)".

$$\lim_{n\to\infty} \bar{X}_n = E[X] \tag{5}$$

Also, S_n^2 is a consistent estimator of σ^2 and this result is represented in Eq. (6)".

$$\lim_{n\to\infty} S_n^2 = \sigma^2 \tag{6}$$

By using the result of "Eq. (6)" in "Eq. (4)", the CLT can now be rewritten as represented in Eq. (7)".

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| \le \varepsilon) = 2\varphi\left(\frac{\varepsilon\sqrt{n}}{S_n}\right) - 1 \tag{7}$$

3.2. *Query Type 1*

SELECT average($R_i.A$)
FROM $R_1, R_2, \dots . R_k$
WHERE $Join(R_1, R_2, \dots R_k)$

Query 3: Query Type 1 (SQL)

Algorithm 3.4. $AQPT_1(Q)$

$list[]$
$k = 1$
$n = 1$
$\bar{X}_n = 0$
$S_n^2 = 0$
$temp = 0$
$est(Q) = 0$
while$(k \le k_{large})$
  $T_s = Random\_Sample\_Generator(Q)$
  $list[k] = R_i.A(T_s)$
  $k++$
end while

$\boldsymbol{for}(k = 1; k \leq k_{large}; k++)$
$\quad \bar{X}_n = \bar{X}_n + list[k]$
$\boldsymbol{end\ for}$
$\bar{X}_n = \frac{\bar{X}_n}{k_{large}}$
$\boldsymbol{for}(k = 1; k \leq k_{large}; n++)$
$\quad S_n^2 = S_n^2 + (list[k] - \bar{X}_n)^2$
$\boldsymbol{end\ for}$
$S_n^2 = \frac{S_n^2}{k_{large}-1}$
$\boldsymbol{while}(n \leq \frac{z_p^2 S_n^2}{\varepsilon^2})$
$\quad T_s = Random\_Sample\_Generator(Q)$
$\quad temp = temp + R_i.A(T_s)$
$\quad n++$
$\boldsymbol{end\ while}$
$est(Q) = \frac{temp}{n}$
$\boldsymbol{Return}\ (est(Q))$

The Query Type 1 -- denoted as Q -- is represented in Query 3, which utilizes the average operation as the aggregate function. The AQPT for Query Type 1 is outlined in Algorithm 3.4., which is denoted as $AQPT_1(Q)$. Initially S_n^2 is calculated. In the first while loop, $Random\_Sample\_Generator()$ is invoked k_{large} (which is ≈ 1000) number of times to generate random samples of $Result(Q)$. Each extracted random sample is denoted as T_s. The attribute value $R_i.A$ corresponding to T_s is denoted as $R_i.A(T_s)$. For each T_s, the corresponding $R_i.A(T_s)$ is stored in the array $list[]$. Using the first and second for loops, $\bar{X}_n$ and S_n^2 are calculated respectively. The final while loop is used for calculating $est(Q)$. This while loop halts after generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of $Result(Q)$ through $Random\_Sample\_Generator()$. Here, z_p denotes $\frac{p+1}{2}$ quantile of the standard normal distribution. For each extracted T_s, the corresponding $R_i.A(T_s)$ is added and stored in $temp$. Finally, $est(Q)$ is calculated as $est(Q) = \frac{temp}{n}$.

Theorem 3.1. *The Algorithm 3.4. achieves the estimation quality outlined in "Eq. (1)".*

Proof.

Let $R_i.A(Result(Q))$ be the random variable denoted as X. In the second while loop, each $R_i.A(T_s)$ generated by $Random\_Sample\_Generator(Q)$ can be considered as random samples of X which will be denoted as $(X_1, X_2, \ldots, X_n)$. Now, $est(Q)$ can be formulated as $est(Q) = \frac{(X_1 + X_2 + \cdots + X_n)}{n} = \bar{X}_n$. Observe that, $F(Q) = \lim_{n\to\infty} \frac{(X_1 + X_2 + \cdots + X_n)}{n}$, because in this scenario, $est(Q)$ will be calculated by uniformly considering all the tuples of $Result(Q)$. By using Law of Large numbers ("Eq. (5)"), if n is very large, then, $F(Q) \approx E[X]$. Since, $\varphi(z_p) = \frac{p+1}{2}$, by substituting, $p = 2\varphi\left(\frac{\varepsilon\sqrt{n}}{S_n}\right) - 1$ in "Eq. (7)", it can be inferred that, by generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X, the estimation quality outlined in "Eq. (7)" can be achieved by Algorithm 3.4. It must be noted that, Algorithm 3.4. halts after generating exactly $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X. Hence, the above Theorem is immediately proved ■

Theorem 3.2. *The Algorithm 3.4. is computationally efficient when $|Result(Q)|$ is large.*

Proof.

Since, $|Result(Q)|$ is large, in both while loops, in every invocation of $Random\_Sample\_Generator(Q)$, the chances of obtaining an empty set is very low. Hence, in the second while loop, the chances of Algorithm 3.4. halting after $\frac{z_p^2 S_n^2}{\varepsilon^2}$ number of invocations of $Random\_Sample\_Generator(Q)$ is very high. It must be noted that, the costliest operation in Algorithm 3.4. is invoking $Random\_Sample\_Generator(Q)$. The number of invocations of $Random\_Sample\_Generator(Q)$ can be bounded as $O(k_{large} + \frac{z_p^2 S_n^2}{\varepsilon^2})$, and it is clear that, this bound is not very large. Further, the number of blocks for each relation of Query 3 is considered large which

implies that, the block sizes are small, and from this it can be inferred that, the Algorithm 3.3. (Reduce Task Invocation Algorithm) requires less computational effort. Hence, the above theorem immediately follows ■

The Theorem 3.1. states that, the Algorithm 3.4. achieves the estimation quality which has been represented in "Eq. (1)". Further, Theorem 3.2. states that, the Algorithm 3.4. is also a computationally efficient technique which makes it practical to be used in real-time scenarios.

3.3. Estimating the Result Set Size

Before outlining the AQPT for Query Type 2, 3 and 4, it is required to describe the Query Result Set Size estimation technique, which will be utilized subsequently.

$$\textbf{SELECT}\ Count(R_i.A)$$
$$\textbf{FROM}\ R_1, R_2, \ldots . R_k$$
$$\textbf{WHERE}\ Join(R_1, R_2, \ldots R_k)$$

Query 4: Query to Calculate Result Set Size (SQL)

The Query 4 -- which will be denoted as Q -- represents the query which is used for calculating the result set size.

Algorithm 3.5. $block\_size\_estimate(Q)$

$\boldsymbol{for}(j = 1; j \leq k; j++)$

$B_{i_j}^j = random\_block\_select(B_1^j, B_2^j, \ldots B_{m_j}^j)$

$assign\_block(Map(B_{i_j}^j))$

$\hat{B}_{i_j}^j = block\_refine(Map(B_{i_j}^j))$

$\boldsymbol{End\ for}$

$N_S = join\_sample\_size(Reduce(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \ldots \hat{B}_{i_k}^k), Q)$

Return (N^S)

Algorithm 3.6. $join\_sample\_size(Reduce(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \ldots \hat{B}_{i_k}^k), Q)$

$\{T_{i_1,i_2,\ldots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\ldots i_k)} = join\_blocks(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \ldots \hat{B}_{i_k}^k, Q)$

$\boldsymbol{If}(\{T_{i_1,i_2,\ldots i_k}^y\}_{y=1,2,..,NJ(i_1,i_2,\ldots i_k)} \neq NULL)$

$N^S = NJ(i_1, i_2, \ldots i_k)$

Return (N^S)

Else

Return $(NULL)$

Algorithm 3.7. $Count(Q)$

$n = 1$

$temp = 0$

$est(Q) = 0$

$\boldsymbol{while}(n \leq n_{large})$

$N_s = block\_size\_estimate(Q)$

$temp = temp + N_s$

$n++$

$\boldsymbol{end\ while}$

$est(Q) = (\prod_{j=1}^k m_j)\frac{temp}{n}$

Return $(est(Q))$

The query result set size estimation technique is outlined in Algorithm 3.7., which is denoted as $Count(Q)$. Here, n represents the number of iterations used. The algorithm halts after generating $n = n_{large}$ steps, where n_{large} is a large positive integer. In each step, the number of result set tuples in a particular set of randomly selected blocks $\in (R_1, R_2, \dots R_k)$ is calculated, and this calculation is performed through $block\_size\_estimate()$, which is described in Algorithm 3.5.

Consider Algorithm 3.5. Here, random blocks are selected uniformly from each $R_j \in (R_1, R_2, \dots R_k)$, and this selection is accomplished through $random\_block\_select()$. Each selected block denoted as $B_{i_j}^j$ is assigned to a unique map task denoted as $Map(B_i^j)$ through $assign\_block()$, which is described in Algorithm 3.2.

In Algorithm 3.5., a reduce task is invoked denoted as $Reduce(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k)$.The reduce task execution algorithm is outlined in Algorithm 3.6. which is denoted as $join\_sample\_size()$. Here all the input blocks $(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k)$ are joined based on the join condition of Q through $join\_blocks()$, in-order to create $\{T_{i_1,i_2,\dots i_k}^y\}_{y=1,2,\dots,NJ(i_1,i_2,\dots i_k)}$. Finally, $NJ(i_1, i_2, \dots i_k)$ is returned.

In Algorithm 3.7., in each iteration, $N_S = NJ(i_1, i_2, \dots i_k)$ is added to $temp$, and after n reaches n_{large} steps, $est(Q)$ is calculated as $est(Q) = (\prod_{j=1}^k m_j)\frac{temp}{n}$.

Theorem 3.3. *The output of Algorithm 3.7. is a consistent estimator of $F(Q)$, which means that,* $\lim_{n\to\infty}((\prod_{j=1}^k m_j)\frac{temp}{n}) = F(Q)$.

Proof.

Let, X denote the number of result set tuples of Q created by the join of any given set of blocks denoted as $(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k)$ based on the join condition of Q. Since, $block\_size\_estimate()$ utilizes uniform random selection of blocks, the output of $block\_size\_estimate()$ can be considered as the uniform random samples of X. Since, $block\_size\_estimate()$ is invoked $n = n_{large}$ number of times, the corresponding outputs will be denoted as $(X_1, X_2, \dots, X_n)$. Let, $u = \prod_{j=1}^k m_j$. By using the Law of Large numbers and the property of uniform random sampling, it is clear that, $u \times \lim_{n\to\infty} \frac{(X_1,X_2,\dots,X_n)}{n} = F(Q)$, which immediately proves the above theorem ■

Theorem 3.4. *The Algorithm 3.7. is computationally efficient.*

Proof.

The Algorithm 3.7. invokes $block\_size\_estimate()$ for $n = n_{large}$ number of times. In each invocation, $(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k)$ are joined based on the join condition specified for Q. Now, m_j is large, and thus, the block sizes are limited. Hence, the join of $(\hat{B}_{i_1}^1, \hat{B}_{i_2}^2, \dots \hat{B}_{i_k}^k)$ requires limited computational effort. Since, in Algorithm 3.7., the main computational cost is the execution of $block\_size\_estimate()$, the above Theorem immediately follows. The Theorem 3.3. states that, the output of Algorithm 3.7. denoted as $est(Q)\ is\ \approx F(Q)$. Hence, $F(Q)$ can be approximated through $est(Q)$. The Theorem 3.4 states that, the Algorithm 3.7. is a computationally efficient technique.

3.4. *Query Type 2*

$$\textbf{SELECT}\ sum(R_i.A)$$
$$\textbf{FROM}\ R_1, R_2, \dots . R_k$$
$$\textbf{WHERE}\ Join(R_1, R_2, \dots R_k)$$

Query 5: Query Type 2 (SQL)

Algorithm 3.8. $AQPT_2(Q)$

$list[]$
$k = 1$
$n = 1$
$\bar{X}_n = 0$
$S_n^2 = 0$
$temp = 0$
$est(Q) = 0$
$\textbf{while}(k \leq k_{large})$
  $T_s = Random\_Sample\_Generator(Q)$
  $list[k] = R_i.A(T_s)$

$k++$
end while
for$(k = 1; k \leq k_{large}; k++)$
$\bar{X}_n = \bar{X}_n + list[k]$
end for
$\bar{X}_n = \frac{\bar{X}_n}{k_{large}}$
for$(k = 1; k \leq k_{large}; n++)$
$S_n^2 = S_n^2 + (list[k] - \bar{X}_n)^2$
end for
$S_n^2 = \frac{S_n^2}{k_{large}-1}$
$N = Count(Q_{count})$
while$(n \leq \frac{z_p^2 S_n^2}{\varepsilon^2})$
$T_s = Random\_Sample\_Generator(Q)$
$temp = temp + N \times R_i.A(T_s)$
$n++$
end while
$est(Q) = \frac{temp}{n}$
Return $(est(Q))$

The Query Type 2 -- denoted as Q -- is represented in Query 5, which utilizes the sum operation as the aggregate function. Also, the count query corresponding to Q which calculates the result set size of Q is denoted as Q_{count}. The Q_{count} is represented in Query 4. The AQPT for Query Type 2 is outlined in Algorithm 3.8., which is denoted as $AQPT_2(Q)$. Initially, S_n^2 and $\bar{X}_n$ are calculated by using the identical procedure outlined in Algorithm 3.4. The algorithm halts after generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of $Result(Q)$ through $Random\_Sample\_Generator()$.The attribute value $R_i.A$ corresponding to T_s is denoted as $R_i.A(T_s)$. For each T_s, the corresponding $R_i.A(T_s)$ is transformed into $N \times R_i.A(T_s)$, added and stored in $temp$. Finally, $est(Q)$ is calculated as $est(Q) = \frac{temp}{n}$.

Theorem 3.5. *The Algorithm 3.8. achieves the estimation quality outlined in "Eq. (1)".*

Proof.

Let $F(Q_{count}) \times R_i.A(Result(Q))$ be the random variable denoted as X. Since, by Theorem 3.3., N is a consistent estimator of $F(Q_{count})$, each $N \times R_i.A(T_s)$ generated by $Random\_Sample\_Generator(Q)$ (in second while loop) can be considered as random samples of X which will be denoted as $(X_1, X_2, \ldots, X_n)$. Now, $est(Q)$ can be formulated as $est(Q) = \frac{(X_1 + X_2 + \cdots + X_n)}{n} = \bar{X}_n$. Observe that, $F(Q) = \lim_{n \to \infty} \frac{(X_1 + X_2 + \cdots + X_n)}{n}$, because in this scenario, $est(Q)$ will be calculated by uniformly considering all the tuples of $Result(Q)$. By using Law of Large numbers ("Eq. (5)"), if n is very large, then, $F(Q) \approx E[X]$. Since, $\varphi(z_p) = \frac{p+1}{2}$, by substituting, $p = 2\varphi\left(\frac{\varepsilon\sqrt{n}}{S_n}\right) - 1$ in "Eq. (7)", it can be inferred that, by generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X, the estimation quality outlined in "Eq. (7)" can be achieved by Algorithm 3.8. It must be noted that, Algorithm 3.8. halts after generating exactly $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X. Hence, the above Theorem is immediately proved ■

Theorem 3.6. *The Algorithm 3.8. is computationally efficient when $|Result(Q)|$ is large.*

Proof.

Let, the cost of executing $AQPT_1()$, $Count()$ and $AQPT_2()$ be denoted as $\text{Cost}(AQPT_1), Cost(Count)$ and $\text{Cost}(AQPT_2)$ respectively. By analyzing the structures of $AQPT_1()$ and $AQPT_2()$, it is clear that, $\text{Cost}(AQPT_2) = O(\text{Cost}(AQPT_1) + Cost(Count))$. By using Theorems 3.2. and 3.4., the above Theorem immediately follows ■

The Theorem 3.5. states that, the Algorithm 3.8. achieves the estimation quality which has been represented in "Eq. (1)". Further, Theorem 3.6. states that, the Algorithm 3.8. is also a computationally efficient technique which makes it practical to be used in real-time scenarios.

3.5. *Query Type 3*

$$\textbf{SELECT}\ variance(R_i.A)$$
$$\textbf{FROM}\ R_1, R_2, \ldots . R_k$$
$$\textbf{WHERE}\ Join(R_1, R_2, \ldots R_k)$$

Query 6: Query Type 3 (SQL)

Algorithm 3.9. $AQPT_3(Q)$

$list[]$
$k = 1$
$n = 1$
$\bar{X}_n = 0$
$S_n^2 = 0$
$temp = 0$
$est(Q) = 0$
$\boldsymbol{while}(k \leq k_{large})$
  $T_s = Random\_Sample\_Generator(Q)$
  $list[k] = R_i.A(T_s)$
  $k++$
$\boldsymbol{end\ while}$
$\boldsymbol{for}(k = 1; k \leq k_{large}; k++)$
  $\bar{X}_n = \bar{X}_n + list[k]$
$\boldsymbol{end\ for}$
$\bar{X}_n = \frac{\bar{X}_n}{k_{large}}$
$\boldsymbol{for}(k = 1; k \leq k_{large}; n++)$
  $S_n^2 = S_n^2 + (list[k] - \bar{X}_n)^2$
$\boldsymbol{end\ for}$
$S_n^2 = \frac{s_n^2}{k_{large}-1}$
$est(Q_{average}) = AQPT_1(Q_{average})$
$\boldsymbol{while}(n \leq \frac{z_p^2 S_n^2}{\varepsilon^2})$
  $T_s = Random\_Sample\_Generator(Q)$
  $temp = temp + (R_i.A(T_s) - est(Q_{average}))^2$
  $n++$
$\boldsymbol{end\ while}$
$est(Q) = \frac{temp}{n}$
Return $(est(Q))$

The Query Type 3 -- denoted as Q -- is represented in Query 6, which utilizes the variance operation as the aggregate function. The $Q_{average}$ is represented in Query 4. The AQPT for Query Type 3 is outlined in Algorithm 3.9. which is denoted as $AQPT_3(Q)$. Initially, S_n^2 and $\bar{X}_n$ are calculated by using the identical procedure outlined in Algorithm 3.4. The algorithm halts after generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of $Result(Q)$ through $Random\_Sample\_Generator()$.The attribute value $R_i.A$ corresponding to T_s is denoted as $R_i.A(T_s)$. For each T_s, the corresponding $R_i.A(T_s)$ is transformed into $(R_i.A(T_s) - est(Q_{average}))^2$, added and stored in $temp$. Finally, $est(Q)$ is calculated as $est(Q) = \frac{temp}{n}$.

Theorem 3.7. *The Algorithm 3.9. achieves the estimation quality outlined in* "Eq. (1)".

Proof.

Let $(est(Q_{average}) - R_i.A(Result(Q)))^2$ be the random variable denoted as X. Since, by Theorem 3.1., $est(Q_{average})$ is a consistent estimator of $F(Q_{average})$, each $(R_i.A(T_s) - est(Q_{average}))^2$generated by $Random\_Sample\_Generator(Q)$ (in second while loop) can be considered as random samples of X which will be denoted as $(X_1, X_2, \ldots, X_n)$. Now, $est(Q)$ can be formulated as $est(Q) = \frac{(X_1 + X_2 + \cdots + X_n)}{n} = \bar{X}_n$. Observe that, $F(Q) = \lim_{n\to\infty} \frac{(X_1 + X_2 + \cdots + X_n)}{n}$, because in this scenario, $est(Q)$ will be calculated by uniformly considering all the tuples of $Result(Q)$. By using Law of Large numbers ("Eq. (5)"), if n is very large, then, $F(Q) \approx E[X]$. Since, $\varphi(z_p) = \frac{p+1}{2}$, by substituting, $p = 2\varphi\left(\frac{\varepsilon\sqrt{n}}{S_n}\right) - 1$ in "Eq. (7)", it can be inferred that, by generating $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X, the estimation quality outlined in "Eq. (7)" can be achieved by Algorithm 3.9. It must be noted that, Algorithm 3.9. halts after generating exactly $n = \frac{z_p^2 S_n^2}{\varepsilon^2}$ random samples of X. Hence, the above Theorem is immediately proved ■

Theorem 3.8. *The Algorithm 3.9. is computationally efficient when $|Result(Q)|$ is large.*

Proof.

Let, the cost of executing $AQPT_1()$ and $AQPT_3()$ be denoted as $\text{Cost}(AQPT_1)$ and $\text{Cost}(AQPT_3)$ respectively. By analyzing the structures of $AQPT_1()$ and $AQPT_3()$, it is clear that, $\text{Cost}(AQPT_3) = O(2 \times \text{Cost}(AQPT_1))$. By using Theorem 3.2., the above Theorem immediately follows ■

The Theorem 3.7. states that, the Algorithm 3.9. achieves the estimation quality which has been represented in "Eq. (1)". Further, Theorem 3.8. states that, the Algorithm 3.9. is also a computationally efficient technique which makes it practical to be used in real-time scenarios.

3.6. *Query Type 4*

```
SELECT STDEV(R_i.A)
FROM R_1, R_2, .... R_k
WHERE Join(R_1, R_2, ... R_k)
```

Query 7: Query Type 4 (SQL)

Algorithm 3.10. $AQPT_4(Q)$

$est(Q) = 0$

$est(Q_{variance}) = AQPT_3(Q_{variance})$

$est(Q) = \sqrt{est(Q_{variance})}$

Return $(est(Q))$

The Query Type 4 -- denoted as Q -- is represented in Query 7, which utilizes the Standard Deviation operation as the aggregate function. The AQPT for Query Type 4 is outlined in Algorithm 3.10. which is denoted as $AQPT_4(Q)$. The $Q_{variance}$ is represented in Query 6. Here, the output of $AQPT_3(Q_{variance})$ is calculated, and which is denoted as $est(Q_{variance})$. Finally, $est(Q)$ is calculated as $est(Q) = \sqrt{est(Q_{variance})}$.

Theorem 3.9. *The Algorithm 3.10. achieves the estimation quality outlined in* "Eq. (1)".

Proof.

It is evident that, $F(Q) = \sqrt{F(Q_{variance})}$. By the proof of Theorem 3.7., $\lim_{n\to\infty} est(Q_{variance}) = F(Q_{variance})$. Consider $F(Q) = h(F(Q_{variance}))$ where $h(F(Q_{variance})) = \sqrt{F(Q_{variance})}$. Thus, $est(Q) = h(est(Q_{variance}))$. Since, $AQPT_3(Q_{variance})$ achieves the estimation quality represented in "Eq. (1)", it can be inferred that, $AQPT_4(Q)$ also achieves the same estimation quality ■

Theorem 3.10. *The Algorithm 3.10 is computationally efficient when $|Result(Q)|$ is large.*

Proof.

Let, the cost of executing $AQPT_4()$ and $AQPT_3()$ be denoted as $\text{Cost}(AQPT_4)$ and $\text{Cost}(AQPT_3)$ respectively. By analyzing the structures of $AQPT_4()$ and $AQPT_3()$, it is clear that, $\text{Cost}(AQPT_4) = O(\text{Cost}(AQPT_3))$. By using Theorem 3.8., the above Theorem immediately follows ■

The Theorem 3.9. states that, the Algorithm 3.10. achieves the estimation quality which has been represented in "Eq. (1)". Further, Theorem 3.10. states that, the Algorithm 3.10. is also a computationally efficient technique which makes it practical to be used in real-time scenarios.

4. Discussions on Empirical Results

4.1. *Experimental Setup Details*

Table 1. Experimental Parameter Setup Values

System Parameters	Used Values
Dataset Size	$3\ TB$
No of Queries used for each Query Type	5
No of Relations in the Join	Varied between $\{3\ to\ 5\}$
m_j	Varied between $\{10^5\ to\ 10^6\}$
p	0.9
ε	Varied between $[10^2\text{- }10^3]$
n_{large}	3×10^3

$$esratio = \frac{|F(Q)-est(Q)|}{F(Q)} \times 100 \tag{8}$$

The experimental setup outlined in [1] is utilized in this empirical analysis study. The experimental parameter setup values are outlined in Table 1. The TPC-DS dataset [1] is utilized for the empirical analysis study. The TPC-DS is a structured dataset. Hence, this dataset is transformed into unstructured format and stored in files. The transformation ensures that, the reverse transformation is feasible and computationally efficient. The performance metric utilized in this empirical study is denoted as $esratio$ [1], and which is represented in "Eq. (8)". It must be noted that, higher and lower values of $esratio$ indicates worser and better performance respectively of the estimator.

The contemporary AQPT outlined in [18] is used in this empirical analysis study. For the ease of reference, the proposed AQPT in this work and the contemporary AQPT [18] will be denoted as PT and CT respectively. The CT executes the given query on a subset of data and estimates the final result. It must be noted that, the CT does not provide any guarantee on estimation error. Specifically, the CT is a fixed sample-size technique in-which, the sample size is fixed.

4.2. *Discussions*

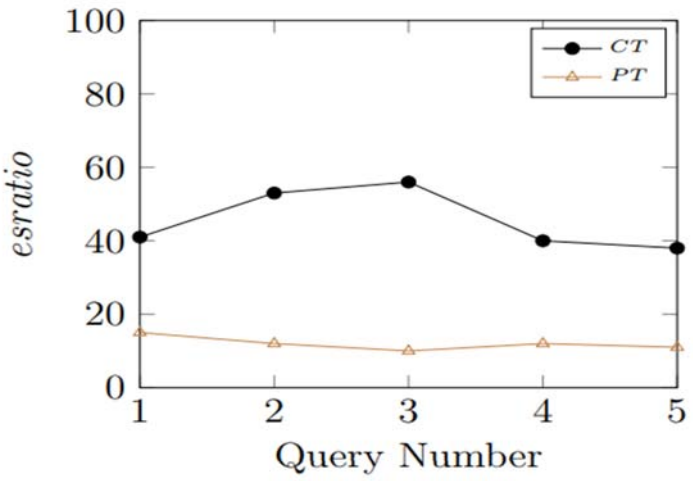

Fig. 1. esratio (Query Type 1)

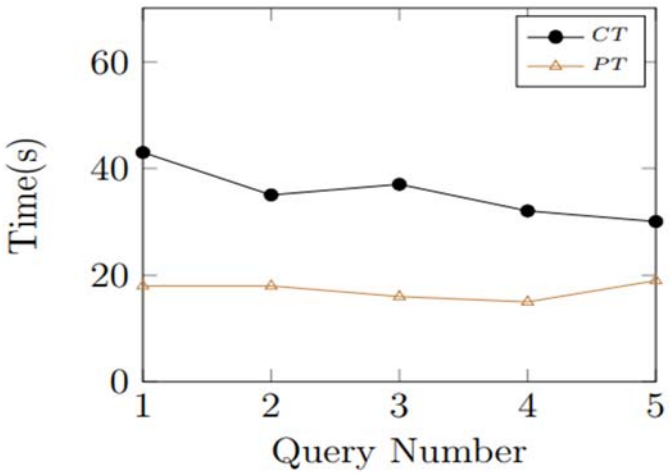

Fig. 2. Execution Latency (Query Type 1)

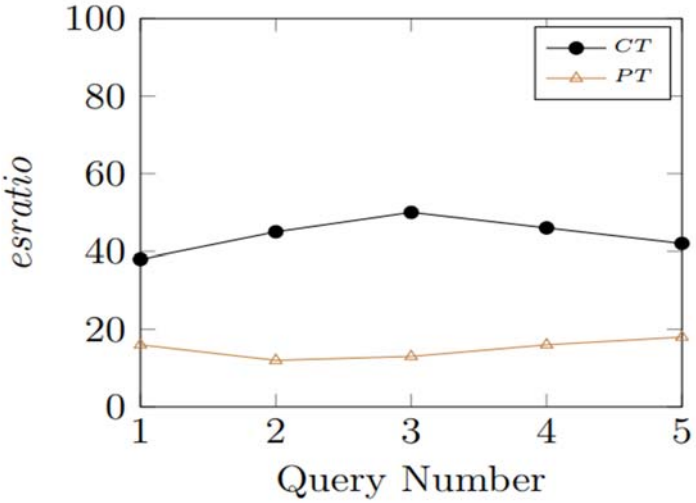

Fig. 3. esratio (Query Type 2)

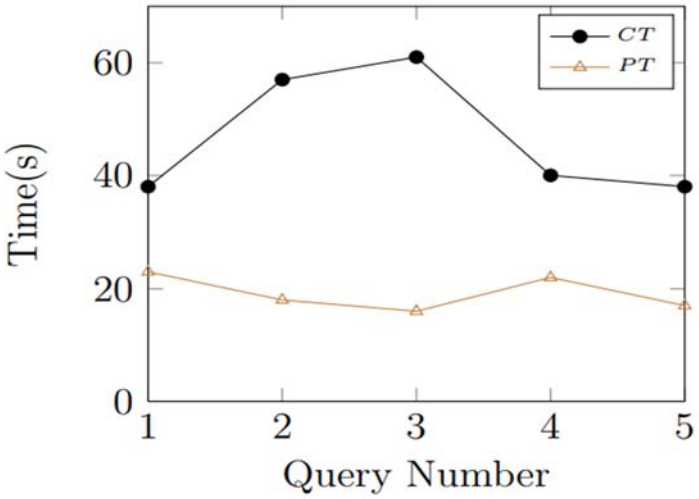

Fig. 4. Execution Latency (Query Type 2)

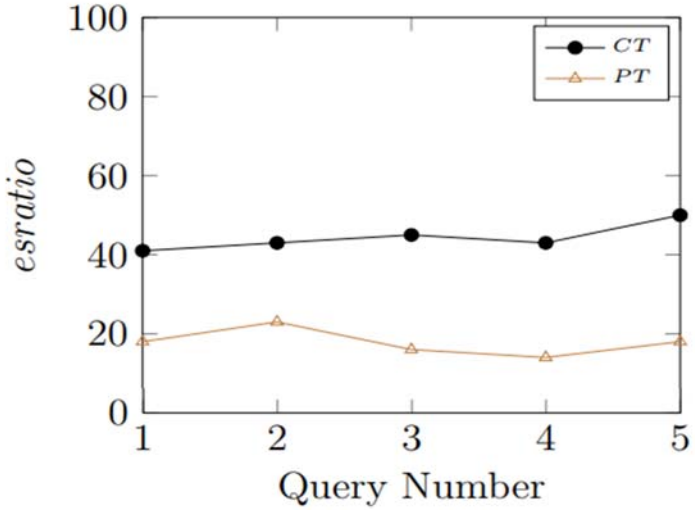

Fig. 5. esratio (Query Type 3)

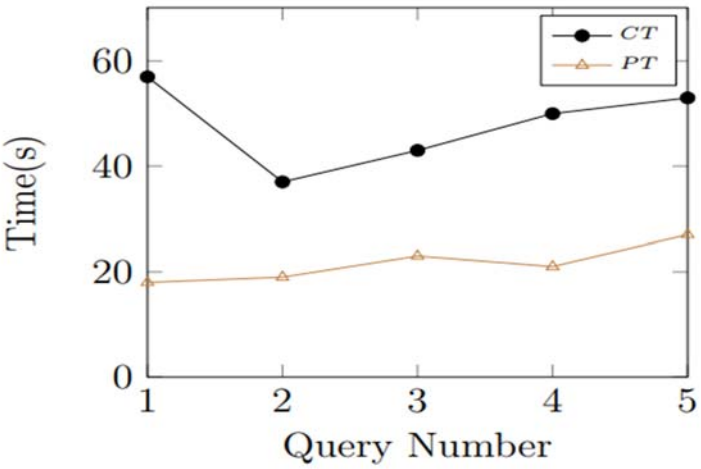

Fig. 6. Execution Latency (Query Type 3)

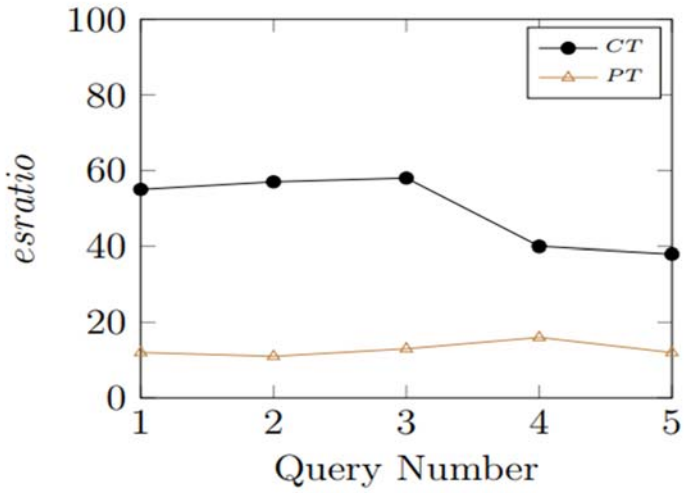

Fig. 7. esratio (Query Type 4)

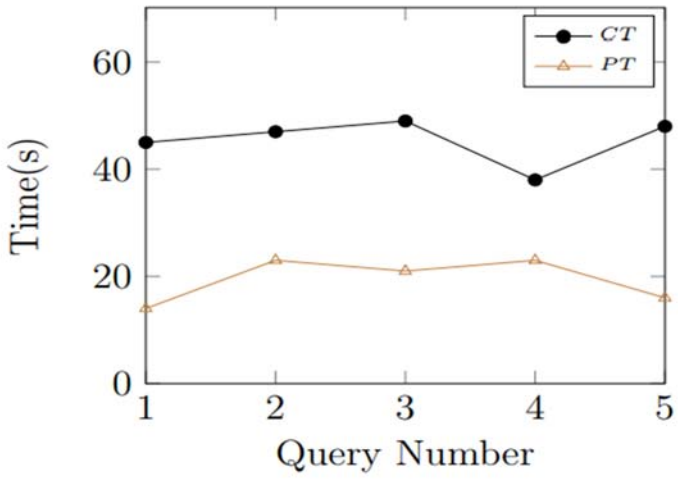

Fig. 8. Execution Latency (Query Type 4)

The first experiment analyzes the performance of *PT* and *CT* against Type 1 Queries by using the *esratio metric*. The result of this experimental analysis is illustrated in Fig. 1. From Fig. 1., it is evident that, *PT* significantly outperforms *CT* in *esratio*. Since, CT is a fixed sample technique, and does not aim to achieve a defined estimation error, it does not scale up to the performance of PT which specifically targets to achieve the predefined estimation error (as outlined in Theorem 3.1).

The first experiment is again analyzed for the performance of *PT* and *CT* against Type 1 Queries by using the *query execution latency metric*. The result of this experimental analysis is illustrated in Fig. 2. From Fig. 2, it is again evident that, *PT* significantly outperforms *CT* in *execution latency*. The CT consumes a greater number of samples in hope of achieving a good estimation accuracy. However, PT achieves its intended goal by consuming relatively number of random samples (as outlined in Theorem 3.2).

The second, third and fourth experiments analyze the performances of *PT* and *CT* against Type 2, 3 and 4 Queries respectively. The results of these experiments are illustrated in figures Fig. 3 through Fig. 8. From these figures, it is evident that, *PT* substantially outperforms *CT* in-terms of both *esratio* and *query execution latency*. The reasoning for the observed performance of *PT* and *CT* in these figures is similar to the reasoning outlined for Fig. 1. and Fig. 2. (refer to Theorems 3.5, 3.6., 3.7., 3.8., 3.9. and 3.10.).

5. Conclusion

The following contributions were presented in this paper: (1) The challenges associated with Big-Data query processing were highlighted. (2) The open issues in the design of AQPT for executing join-aggregate queries on Big-Data were described. (3) To address the outlined open issues, AQPT was presented which specifically addressed four aggregate operations -- average, sum, variance and standard deviation, and this proposed AQPT was designed by utilizing CLT (4) Theoretical bounds WRT sample size consumed by the proposed AQPT was presented. (5) Empirical analysis study of the proposed AQPT and its contemporary technique was presented. In this study, the proposed AQPT substantially outperformed the contemporary technique in-terms of result estimation quality and computational efficiency.

It must be noted that, the sampling techniques are practically feasible when the number qualifying tuples for the given query is considerably large, and this information is not available before the query is executed. Hence, in future, we would investigate on designing a framework which would inform the feasibility of utilizing AQPT for executing a given join-aggregate query.

References

[1] Malatesh S Havanur and Y. S. Kumaraswamy. (2018). Efficient Execution of Aggregate Queries on Big Data. International Journal of Applied Engineering Research.
[2] Malatesh S Havanur and Y. S. Kumaraswamy. (2018). Approximation Techniques for Execution of Aggregate Queries on Big Data. International Journal of Engineering Science and Technology (IJEST).
[3] J.M. Hellerstein, P.J. Haas and H.J. Wang. (1997). Online Aggregation. In: Proceedings of ACM SIGMOD Conference.
[4] C.M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. (2007). Scalable Approximate Query Processing with the DBO Engine. In: Proceedings of SIGMOD Conference.
[5] M.L. Kersten, S. Idroes, S. Manegold and E. Liarou. (2011). The Researcher's Guide to the Data Deluge: Querying a Scientific Database in just a few Seconds. In: PVLDB.
[6] S. Chaudhari, V.R. Narasayya and R. Ramamurthy. (2011). Estimating Progress of Execution for SQL Queries. In: Proceedings of SIGMOD Conference.
[7] G. Luo, J. F. Naughton, C.J. Ellman and M. Watzke. (2004). Toward a Progress Indicator for Database Queries. In: Proceedings of SIGMOD Conference.

[8] A. Ganapathi, H. A Kuno, U. Dayal, J.L.Weinter, A.Fox, M.I. Jordan and D.A. Patterson. (2009). Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In: Proceedings of International Conference on Date Engineering.
[9] M.Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. Zdonik. (2012). Learning based Query Performance Modeling and Prediction. In: Proceedings of International Conference on Date Engineering.
[10] Y.E. Ioannidis. (2003). The History Histograms (abridged). In: Proceedings of VLDB Conference.
[11] R.J. Lipton and J.F. Naughton. (1990). Query Size Estimation by Adaptive Sampling. In: Proceedings of PODS Conference.
[12] P.J. Haas, J.F. Naughton, S. Sheshadri, and A.N. Swami. (1996). Selectivity and Cost Estimation for Joins based on Random Sampling. In: J. Comput. Syst. Sci.
[13] P.J. Haas, J.F. Naughton, S. Seshadri and L. Stokes. (1995). Sampling-based Estimation of the Number of Distinct Values of an Attribute. In: Proceedings of VLDB Conference.
[14] P.J. Haas and A.N. Swami. (1992). Sequential Sampling Procedures for Query Size Estimation. In: Proceedings of SIGMOD Conference.
[15] S. Chaudhuri, R. Motwani, and V.R. Narasayya. (1999). On Random Sampling over Joins. In: Proceedings of SIGMOD Conference.
[16] M. Charikar, S. Chaudhuri, R. Motwani, and V.R. Narasayya. (2000). Towards Estimation Error Guarantees for Distinct Values. In: Proceedings of PODS Conference.
[17] Raman Grover and Michael J. Carey. (2007). Extending Map Reduce for Efficient Predicate Based Sampling. In: Technical Report.
[18] V.Ayyalasomayajula. (2011). A Heterogeneous Engine for Running Data-Intensive Experiments and Reports. M.S. Thesis University of California-Irvine.
[19] S. Babu. (2010). Towards Automatic Optimization of Map-Reduce Programs. In: Proceedings of SoCC Conference.
[20] J. Dean and S. Ghemaeat. (2004). Map Reduce. Simplified Data Processing on Large Clusters. In: Proceedings of OSDI Conference.
[21] C.Olston and B.Reed et.al. (2008). Pig-Latin: A not-so-Foreign Language for Data Processing. In: Proceedings of SIGMOD Conference.
[22] A. Thusoo and Z. Shao et al. (2010). Data Warehousing and Analytics Infrastructure at Facebook. In: Proceedings of SIGMOD Conference.