AQPrius: Offline Approximate Query Processing Enhanced by Error Assessment using Bootstrap Sampling

Feng Yu *fyu@ysu.edu*
Computer Science and Information Systems
Youngstown State University
Youngstown, OH 44555, USA

Sabin Maharjan *smaharjan02@student.ysu.edu*
Computer Science and Information Systems
Youngstown State University
Youngstown, OH 44555, USA

Lucy Kerns *xlu@ysu.edu*
Statistics and Mathematics
Youngstown State University
Youngstown, OH 44555, USA

Xiangjia Min *xmin@ysu.edu*
Bioinformatics and Plant Biology
Youngstown State University
Youngstown, OH 44555, USA

Abdu Arslanyilmaz *aarslanyilmaz@ysu.edu*
Computer Science and Information Systems
Youngstown State University
Youngstown, OH 44555, USA

Michelle Zhu *zhumi@montclair.edu*
School of Computing
Montclair State University
Montclair, NJ 07043, USA

Abstract

In this work, we present AQPrius, an offline approximate query processing (AQP) engine that can efficiently answer complex analytic queries on large datasets. Unlike existing systems that employ the online AQP schemes, AQPrius employs the offline AQP scheme which has two advantages: (1) it doesn't require high-end hardware or expensive auxiliary data structures such as indices or hash tables; (2) the synopses collected are reusable for future queries on the same database which can significantly save computing resources. However, the error assessment for offline AQP systems is still a challenging problem. The contributions of this research are four-fold. First, AQPrius is an offline AQP engine that can quickly answercommon analytic queries including selection conditions, join conditions, and aggregate functions. It can speed up complex query processing on big data. Second, AQPrius enables error assessment using a non-parametric statistic method, namely bootstrap sampling, that can provide the standard error of query estimation. Third, using the standard error by bootstrap sampling, we extend the traditional offline AQP system from providing a single-point query estimation to a range estimation which is a bounded answer presented as a confidence interval (CI). Finally, the system is developed using the Rust programming language which can prevent many security issues and potential vulnerabilities. We evaluate AQPrius using the well-known TPC-H benchmarks. The experimental results show that AQPrius can rapidly generate accurate bounded query answers for various test queries with selection and join conditions.

Keywords: Approximate Query Processing, Bootstrap Sampling, Big Data.

1. INTRODUCTION

Efficiently processing complex queries has always been a challenge for big data management systems. Much work has been focused on promptly executing data queries using advanced hardware and software solutions (D. L. Quoc *et al.*, 2018; M. Sch *et al.*, 2015). One fact ignored by many big data management systems is that computing the exact query answer can be expensive and unnecessary in many scenarios. For example, one user may only desire quickapproximations to some testing queries at the beginning of an exploratory data analysis (or EDA) project. This stimulates the study of fast query estimation on big data (J. Bater, 2020; T. Tian, 2020; D. Wilson *et al.*, 2019; Z. Zhou *et al.*, 2018; T. Siddiqui *et al.*, 2020).

Approximate query processing (or AQP) is an alternative scheme to answer queries approximately with satisfying accuracy and within a short time (S. Agarwal *et al.*, 2014; K. Li *et al.*, 2018; Q. Liu, 2009). Compared with traditional query processing techniques, AQP doesn't need to execute a query on the original data which can be very costive. Instead, it usually collects statistic summaries, named synopses, from the original data and runs the query on the synopses to produce a synopsis query result which can be substantiallyfaster and less costive. The synopsis query result will later be used by AQP statistic estimators to produce query estimations.

Depending on how the synopses are collected, AQP schemes can be categorized into the *online AQP* and the *offline AQP (*S. Chaudhuri *et al.*, 2017). The online AQP (Y. Chen *et al.*, 2017; V. Leis *et al.*, 2017; F. Li *et al.*, 2019), by the name, collects statistic synopses on the flywhen the user query for approximation is submitted. The procedure of synopsis collection can be time-consuming given a large dataset. Therefore, online AQP techniques usually rely on auxiliary and expensive data structures, such as indices and hash tables, to collect statistics quickly. These data structures can be either costly to build or expensive to maintain on big data. Another drawback of online AQP schemes is that the collected statistics are not reusable and must be re-collected for estimation by a different user query.

Opposite to online AQP, offline AQP (S. Acharya *et al.*, 1999; F. Yu *et al.*, 2013) collects statistic synopses before a user query is submitted. This technique usually needs knowledge of the schema of a database to summarize it into smaller synopses. Offline AQP doesn't require auxiliary data structures to collect statistics because the synopses are already available when a user query is submitted. This avoids the extra time of synopses collection and will not delay the query estimation during the system runtime. One challenge for offline AQP systems is to efficiently assess the errorofquery estimation.

The common applications of AQP include estimation for selection queries (or selection-AQP or σ-AQP) and multi-table join queries (or join-AQP, $\bowtie$-AQP). The challenge is that when query conditions change the underlying distributions of the query result sets will also change and are hard to predict. For online AQP schemes, because the synopses are collected before a query is submitted, the distribution of the results can be calculated. Therefore, the variance of online AQP estimators can be computed. However, for offline AQP schemes, it's hard to predict the distribution of the query result set before a query is submitted. Therefore, the error assessment of offline AQP schemes can be challenging.

To this end, we propose AQPrius, a sampling-based offline AQP system that not only can provide users with accurate query estimation at lower costs but also can provide error assessment for query estimation. This system uses the offline AQP technique to collect synopses from the original database which is suitable to run on resource-constraint platforms. The system includes a general database connector that can communicate with various data sources such as centralized or distributed databases. AQPrius can take common user queries including selection and join queries with various query conditions and analytic functions. This system provides users with fast and satisfying query answers.

A unique feature of this system is it employs bootstrap sampling (B. Efron *et al.*, 1994) which is a special statistical method that can assess the errors of query estimations of offline AQP. An advantage of bootstrap sampling is that it doesn't require prior knowledge of the data distribution

but can work like “pull itself up by its bootstrap”. Bootstrap sampling performs a special sampling method, named resampling, which generates many random samples with replacement, named bootstrap samples, from the original sample. By applying the statistic estimator on the bootstrap samples, a set of scalar values of estimation, named bootstrap replications, is obtained. The standard error of the bootstrap replications can be computed as the error assessment of the statistic estimator.

The AQPrius system consists of three major components including a query processing engine, a synopses engine, and a bootstrap engine. The query processing engine is the system manager coordinating other components in the system. It has a query parser that can read and analyze common user queries with various query conditions. The synopses engine can access external databases to collect statistic synopses. This system employs the CS2 (F. Yu *et al.*, 2013) which is an efficient offline AQP scheme capable of answering common analytic queries while only taking tiny storage. The collected synopses will be reduced in size to fit into a centralized storage. The bootstrap engine utilizes the intermediate results from the query processing engine for error assessment. It will provide a standard error and a confidence interval to enrich the user feedback.

The contributions of this work are listed as follows.

1. AQPrius is a practical offline AQP framework that answers common queries including selection and join conditions with aggregate functions. It is capable of systematically reducing big data into smaller-sized statistic synopses. The collected synopses can be stored in centralized storage systems such as hard drives or main memory to speed up the query estimation time.
2. This system extends the existing offline AQP systems by introducing the error assessment functionality. It employs bootstrap sampling, a non-parametric statistic method to efficiently estimate the standard error of query estimations just in time. Bootstrap sampling doesn't need prior knowledge of the query result set distributions and can provide an accurate standard error for query estimation.
3. The system can not only provide a point estimation (or single value estimation) of the query result; but can also extend the offline AQP by providing a range estimation (or confidence interval, CI) which enriches the user feedback. The CI will provide a user with upper and lower bounds of the query estimation to better help the decision-making process.
4. The system is developed using Rust which is a popular systematic programming language. Rust is designed to prevent to the maximum extent common security bugs and vulnerability issues, such as unsafe pointers and memory leaks. In addition, Rust enables fast runtime performance comparable to traditional systematic programming languages such as C or C++. The system is designed in self-contained modules which makes future extensions of the system easier. The source code of the system is available at: https://github.com/YSU-Data-Lab/aqprius.

The rest of the work is organized as follows. Section 2 introduces the background of query estimation and bootstrap sampling. Section 3 presents the overview of the developed system. Section 4 formulates the framework of query estimation. Section 5 describes the error assessment framework. Section 6 presents the system design. The experiment results are presented in Section 7. Section 8 includes the related work. Section 9 states the conclusion and future work.

2. BACKGROUND

2.1 Join Graph

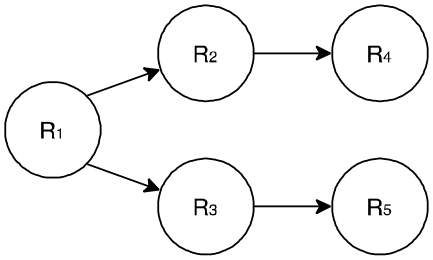


FIGURE1: An Example Join Graph.

Definition 1.(Join Graph) A join graph is a visual representation of a database in which the flow of joins is depicted. The tables of a database are depicted as circles or nodes. The join graph can also entail information on types of join relationships such as many-to-many, many-to-one, and one-to-one. The joinable relationships are depicted as edges between the nodes. The edge with an arrow denotes a many-to-one relationship where the arrow points to the "one" side of the relationship.

Definition 2. (Joinable Relation) Two relations, namely R_i and R_k, $i \neq k$, are considered joinable relations, when there exists a connected path between them with a length greater than 1.

Definition 3. (Joinable Tuple) Assume that R_i and R_k are joinable relations. A tuple in R_i, denoted by t_i, and a tuple in R_k, denoted by t_k are considered joinable tuples if t_i can match a tuple t_{i+1} in R_{i+1}, t_{i+1} can match a tuple t_{i+2} in R_{i+2}, …, and t_{k-1} can match t_k in R_{i+1}. In this context, the matching is to associate two tuples following the joinable relations between the tables.

Figure 1 depicts an example join graph of a database including five tables R_1, R_2,…, and R_5. As shown in the figure, table R_1 is joinable with tables R_2 and R_3. Table R_2 is joinable with Table R_4, but it's not joinable with R_3 or R_5.Table R_3 is joinable with table R_5, but it's not joinable with R_2 or R_4. In addition, $R_i \rightarrow R_j$ denotes a many-to-one or one-to-one join relation from R_i to R_j, orR_j is on the one side.

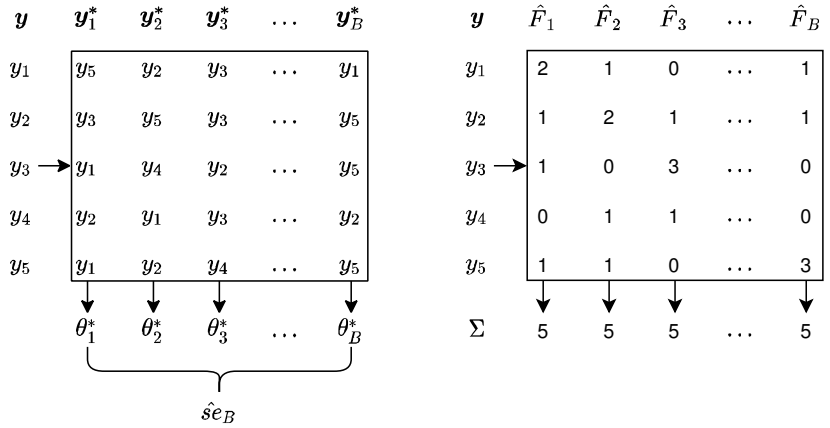
2.2 Correlated Sampling

Simple random sampling was originally introduced as a synopsis only for simple queries such as selection. However, simple random sampling is not suitable to answer more complex queries, such as join queries, because the joinable tuples between relations will introduce correlation which is not retained by sample random sampling across multiple tables.

To this end, correlated sampling is introduced to retain the joinable relationship between sample tuples. Both JS (S. Acharya *et al.*, 1999) and CS2 (F. Yu *et al.*, 2013) employ correlated sampling to generate sample-based synopses for AQP purposes. The first step starts from a simple random sampling without replacement (SRSOWR) on a relation, namely R_1, where random sample S_1^* is collected. The next joinable relation with R_1, denoted by R_2, will be sampled. To retain the correlation between R_1 and R_2, instead of randomly sampling R_2, the joinable tuples of R_1 with S_1^* are collected. This procedure will continue if there is another joinable relation.

Even though both JS and CS2 employ correlated sampling, their generated synopses are distinct. JS requires performing SRSOWR on each relation followed by correlated sampling in the join graph of a database. This may lead to high sampling costs in a database with a complex join graph. However, CS2 doesn't need SRSOWR on each relation; it only needs the relations on top of the paths in a join graph named source relations. There can be multiple source relations in a join graph. By SRSOWR the source relations and correlated sampling on the joinable relations with the joinable relations, the CS2 synopses can retrieve all needed tuples for join query estimations. A special case is, that when some join queries do not include the source relation,

named no-source join queries, there can be bias in the query estimation. To address this, the CS2 framework introduced the JR value and RV estimator which doesn't need additional simple random samples from the database and can produce unbiased estimations for non-source join queries.



(a) Bootstrap Resamples (b) Bootstrap Distributions

FIGURE 2: Bootstrap Sampling Example.

2.3 Bootstrap Sampling

Bootstrap sampling (B. Efron *et al.*, 1994) is a well-known non-parametric method with versatile applications in statistical analysis. A popular application of bootstrap sampling is for the error assessment of a statistic estimation based on a given sample when there is no prior knowledge of the source data distribution.

Bootstrap sampling uses a unique procedure named resampling or simple random sampling with replacement (SRSWR). Each resampling from the original sampled data generates a new distribution named *bootstrap sample*. The desired statistic estimator or analytic function is then applied to the bootstrap sample to compute a scalar value named *bootstrap replication*. Bootstrap sampling usually generates a large number of bootstrap replications (typically between 1000 to 2000) to estimate statistical features, such as the standard deviation of the statistic estimator, even when the population (or ground truth) distribution is unknown.

Figure 2 depicts a simple example of the bootstrap resampling procedure. The original sample dataset $\vec{y} = (y_i)_{i=1}^n$ is obtained from a population with unknown distribution F. A bootstrap sample $\vec{y}^* = (y_i^*)_{i=1}^n$ is obtained by performing SRSWR on$\vec{y}$ for n times. For example, if n=5, there can be multiple possibilities of SRSWR on $\vec{y}$, such as $\vec{y}_1^* = (y_5, y_3, y_1, y_2, y_1)$, $\vec{y}_2^* = (y_2, y_5, y_4, y_1, y_2)$, $\vec{y}_3^* = (y_3, y_3, y_2, y_3, y_4)$, etc. These resamples are depicted in Figure2 (a). Applying a statistic estimator on each $\vec{y}_i^*$ generates a bootstrap replication θ_i^*, $i = 1, \dots, B$. The standard error of all θ_i^*, denoted by $\widehat{se}_B$, will be used to estimate the standard error of the statistic estimator.

The distribution of each bootstrap sample $\hat{F} = (\hat{f}_1, \hat{f}_2, \dots)$ can be obtained by summarizing the frequency of each sampled element, where $\hat{f}_k = \#\{y_i^* = y_k\}/n$. Figure 2(b) depicts the distributions of bootstrap samples in this example.

3. SYSTEM OVERVIEW

Figure 3depicts the overview of the developed system AQPrius. It extends the capabilities of existing AQP systems in three aspects: (1) It includes an offline AQP synopses engine for complex query estimations including join queries. It only needs to collect synopses once in an offline manner and can be used to estimate all future queries following the join graph of the database. (2) It includes an error assessment component that can rapidly evaluate the errors

produced from AQP query estimations. The error assessment module employs a nonparametric statistic method named bootstrap sampling to produce an unbiased estimation of the standard deviation of a query estimation result. (3) It extends the original offline AQP system from providing a point estimation (or a single value) to a range estimation (or a bounded answer) when doing query estimation. The bounded answer will be presented as a confidence interval (CI) to a user to enhance the usability of the system.

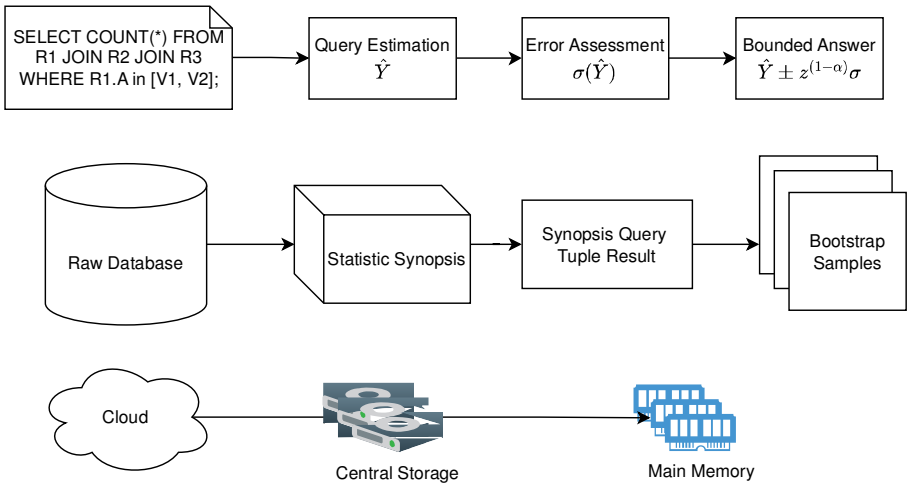


FIGURE3: System Overview.

There are three layers of workflows entailed in the system: (1) the query processing workflow: When a complex query is submitted by a user, the system will parse the query and produce a query estimation. After that, the error assessment will be performed to evaluate the accuracy of the query estimation. Finally, a bounded answer will be presented to the user. (2) the synopses management workflow: Before any query is given, the system will produce offline statistic synopses from the original raw database. This will reduce the big data into smaller synopses. The statistic synopses will produce tuple results during the query estimation. The tuple results will be employed by the bootstrap sampling to produce an error assessment for query estimation. (3) the data storage workflow: The original database may contain big data and be stored in the cloud or a big data cluster. The system will reduce the original big data into small statistic synopses and can be stored on a centralized storage server. Finally, the tuple results and bootstrap resamples are created which will be further reduced to fit into the main memory or the device memory of a computing accelerator, such as a GPU. In general, these three layers will collaborate to achieve the designed functionalities of the system.

3.1 Query Formulation

The developed system supports common analytic SQL queries. Compared with other systems, it can accept more complex query operators, such as a selection with multiple conditions and a multi-table join, which are widely used in business databases. It also supports common data analytic operators. The query pattern, Q, supported in this system is formulated as follows.

```
SELECT Fn(attribute collection) FROM table collection WHERE query conditions;
```

where Fn is a common and "smooth" aggregate function such as COUNT, AVG, and SUM. It's well-known that non-smooth aggregate functions, for example, MEDIAN and COUNT DISTINCT, are not suitable for sample-based AQP frameworks; the table collectionincludes all tables involved in the query (denoted by $R^Q = \{R_i^Q\}_{i=1}^m$); the attribute collection includes the attributes needed by the query result (denoted by $A^Q = \{A_{ij}^Q\}$); the conditions in the WHERE clause can include both selection conditions and join conditions. Particularly, the selection conditions will filter which results will be included and the join conditions will decide how multiple tables are

associated. The system does not limit the number of selection conditions and join conditions in the query.

4. QUERY ESTIMATION

The estimation of query results is formulated in this section. We divide the scenarios into two parts including the estimation for selection queries and the estimation for multi-table join queries. The estimation principles and estimator formulas are presented.

4.1 Selection Query Estimation

When a query Q only includes SELECTION operations, we perform query estimation based on the query result on the sample table S. Each sample tuple $u_i \in S$ will produce a tuple query result y_i based on the aggregate function in the SELECT clause. For example, if the aggregate function is COUNT, then y_i will be either 1 if the sample tuple u_i satisfies the selection or 0 otherwise. The query result Y_s on the sample table S is calculated as $Y_s = \sum_{i=1}^{n} y_i$ where $n = |S|$ is the sample size. Suppose the size of the original table R is N, and the sample fraction $f = \frac{n}{N}$, then the estimation of the ground truth query result is $\hat{Y} = \frac{Y_S}{f}$.

4.2 Multi-Join Query Estimation

When a query Q includes JOIN operations, we perform query estimation using the query result on the sample tables in a synopsis $\mathbb{S}$. Suppose the synopsis $\mathbb{S}$ is constructed on the original dataset $R = \{R_i\}_{i=1}^{M}$. The top relation in the join graph of query, denoted by R^Q, is R_1^Q and its corresponding sample table is S_1^Q. When a query Q is executed on the synopsis $\mathbb{S}$, each sample tuple $u_i$in S_1^Q, $i = 1 \ldots n$, will produce a tuple query result y_i based on the query conditions. Let $S_Q = \{y_i\}_{i=1}^{n}$ denote the collection of tuple query results. For instance, if the aggregate function is COUNT, then each y_i will be either 1 if the tuple u_i satisfies the query conditions or 0 otherwise.

The synopsis query resultY_s of Q can be calculated as $Y_s = \sum_{i=1}^{n_1} y_i$ where n_1 is the size of S_1^Q or $n_1 = |S_1^Q|$. Suppose $\mathrm{N}_1 = |\mathrm{R}_1^{\mathrm{Q}}|$ and the sample fraction $f = \frac{n_1}{N_1}$, then the estimation of the ground truth query result of Q, denoted by Y_{GT}, is $\hat{Y} = \frac{Y_S}{f}$.

The estimation of the query doesn't include the top relation of the synopses are named no-source queries (F. Yu *et al.*, 2013). Estimations of no-source queries will need additional data structures such as JR values and RV estimators to produce unbiased estimation. The current system doesn't consider these special join queries. Estimation of these queries will be implemented in later versions of the system.

5. ERROR ASSESSMENT

The system implements the error assessment of query estimation by employing a non-parametric method named bootstrap sampling. The bootstrap sampling performs many resamples on the synopsis results generated during the query estimation procedure. These resamples can help to estimate the standard deviation of the query estimation.

5.1 Bootstrap Sampling

Without loss of generality, we take the error assessment for join query estimation as an example. The error assessment for selection query estimation is a simplified case compared with the join query estimation.

During the estimation of a join query Q, there are tuple query results $S_Q = \{y_i\}_{i=1}^{n}$ obtained on the synopsis $\mathbb{S}$. We perform bootstrap resampling on the S_Q for a total of B iterations. In each iteration, S_Q is uniformly sampled with replacement for n times. For example, $\vec{y}_j = \{y_{j,i}\}_{i=1}^{n}$ is a bootstrap resample where each $y_{j,i}$ is uniformly sampled with replacement from S_Q. A query result based on the bootstrap resample $Y_{\vec{y}_j}$ can be obtained when applying the same query estimator

using $\vec{y}_j$. The query estimation $\hat{Y}_j$ can be calculated as $\hat{Y}_j = \frac{Y_{\vec{y}_j}}{f}$ where f is the sample fraction. For instance, if the aggregate function is COUNT, the estimation $\hat{Y}_j = \frac{1}{f}\sum_{i=1}^{n} y_{j,i}$. $\hat{Y}_j$ is named a bootstrap replication of the query estimation.

The collection of all B bootstrap replications is denoted by $\hat{Y}_B = \{\hat{Y}_j\}_{j=1}^{B}$. The standard error of $\hat{Y}_B$ is

$$\widehat{se}_B = \left[\frac{1}{B-1}\sum_{j=1}^{B}\left(\hat{Y}_j - \bar{\hat{Y}}_B\right)^2\right]^{\frac{1}{2}} \quad (1)$$

where $\bar{\hat{Y}}_B$ is the mean of all bootstrap replications $\hat{Y}_B$. By the theory of bootstrap sampling, Eq (1) is the bootstrap estimation for the standard error of $\hat{Y}$ which is the estimation of the ground truth query result Y_{GT}. The accuracy of $\widehat{se}_B$ increases as the value of B becomes larger. In practice, setting the value of B between 1000 to 2000 is considered as sufficient.

5.2 Bootstrap Confidence Interval (CI)

A confidence interval (CI) with a lower bound and upper bound is a range estimation of the ground truth query result. There are various schemes to compute the CI using bootstrap sampling, such as the standard method and the percentile method. These methods are usually efficient and quite accurate. Improved methods are also available, such as BCA and ABC, which can increase the estimation accuracy and reduce the bias. However, these methods often have slower performance when applied to large datasets (B. Efron *et al.*, 1994). The developed system employs the commonly used standard method which is a generally applied method.

We elaborate on how the standard bootstrap CI is calculated. Let a probabilistic value α be the significant level of CI. The value of α is commonly set to 5% for the 90% level of confidence and 2.5% for the 95% level of confidence, respectively. The bootstrap CI calculated by the standard method is as follows.

$$\left(\hat{Y} - z^{(1-\alpha)} \cdot \widehat{se}_B, \hat{Y} + z^{(1-\alpha)} \cdot \widehat{se}_B\right) \quad (2)$$

where $\hat{Y}$ is the query estimation and $z^{(1-\alpha)}$ is the $100(1-\alpha)$th percentile of a standard normal distribution. For example, for the 90% level of confidence, $z^{(0.95)}$=1.645, and for the 95% level of confidence, $z^{(0.975)}$=1.960.

6. SYSTEM IMPLEMENTATION

6.1 System Architecture

Figure 4 depicts the system architecture of the developed system which consists of three major modules including a query estimation module, an error assessment module, and a query processing model. Among them, the query processing model coordinates the other two modules and keeps the system running. The system can produce three categories of information for user feedback including the ground truth query result, the estimated query result, and the bounded answer for query estimation, namely the confidence interval of the query estimation.

The *query processing module* stands in the center of the system. Once a query is submitted to the system. The query parser will analyze the query statement and collect the necessary information later used by the query estimation module. The information includes tables in the query, selection conditions, join conditions, analytic functions, and result attributes. They will be then transferred to the query estimation module for query estimation.

The *query estimation module* consists of three parts including a database connector, a synopsis engine, and a synopsis store. This module also enables three functionalities including synopsis creation, query estimation, and ground truth query computation.

(1) For synopsis creation, the database connector contacts an external database system or data API to retrieve the schema information and sampled data tuples. The schema information includes the table schema information (or data dictionary) and the relationships between tables (such as joinable relationships). The synopsis engine creates the statistic synopses for the usage of query estimation in an offline manner. It needs to explore the joinable relationships of the database during the synopses creation procedure. The produced synopsis will be stored in the synopsis store. To achieve high speed of query estimation, the system usually employs a smaller sampling fraction to produce synopses small enough to fit into a centralized storage.
(2) During the query estimation procedure, the parsed query statement will be executed on the synopsis store first to collect the intermedia query results, namely the synopsis tuple results. The query execution engine will collaborate with the synopsis store in this process. Depending on the type of the submitted query, they will be used by various query estimators coded in the system to produce query estimations. The synopsis tuple results will be retained in the query execution engine for error assessment.
(3) If the ground truth query result is requested by the user, the query processing module will connect to the original database to compute the ground truth query result.

The*error assessment module* includes two parts including the bootstrap engine and the bootstrap sample store. During the error assessment procedure, the error assessment module will retrieve the synopsis tuple results from the query estimation engine. Many iterations of bootstrap resampling will be conducted using the synopsis tuple results to derive the standard deviation of the query estimation which will be used by the error assessment. To reduce storage overheads, only one collection of bootstrap resamples will be maintained in each iteration. Finally, a bounded answer, or a confidence interval, of the query estimation will be computed based on the standard deviation obtained from bootstrap resampling, and it will be provided to the user.

6.2 System Implementation

The AQPrius system is developed using the Rust programming language. Rust has becomepopular for systems programming. Traditional operating and database systems are predominately developed in C, C++, or Java. As the system becomes more complex, manysoftware issues arise such as memory leaks and vulnerabilities. To address these issues, Rust has introduced many new features to its compiler and toolkits. The components of the developed system are built using Rust which can significantly limit the possibility of producing vulnerabilities and bugs.

The data storage and synopsis store can utilize external database systems. The system has a database connector that can connect to various database systems. The current system is implemented using SQLite as a backend data store. SQLite is a simple, fast, and versatile database system that can reach high throughput of data access on centralized storage. It serves as the central data storage for synopsis creation and reading.

AQPrius can be used across multiple platforms or operating systems. A user can run the system using the command line interface (CLI) in any operating system that supports the Rust programming language.

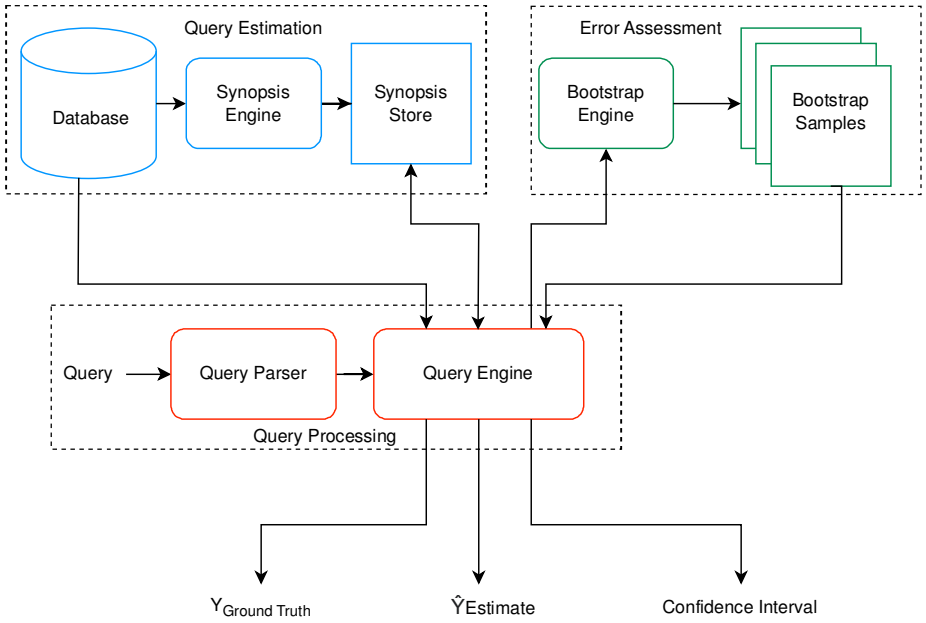


FIGURE4: System Architecture.

6.3 System Runtime

Figure 5 depicts a demo of the system runtime. The system parameters for the demo include sample fraction as 1.0% and total bootstrap iterations as 2000. For a small-scale demo, the source database employed is TPC-H 100MB. The test query for estimation is as follows:

```
select count (*) from lineitem, orders, customer where l_orderkey = o_orderkey
      and o_custkey = c_custkey and c_acctbal< 10000 and c_acctbal> 5000
```

The test query is a three-table join query with selections. The system can parse the query and collect the necessary information for query estimation. The tables in the query include lineitem, orders, and customer from the TPC-H benchmark. The join conditions and selection conditions are analyzed and displayed in a format similar to JSON. The demo generated the synopses needed for query estimation and displayed the operation information for user feedback.

The user feedback includes the database ground truth result, sample query estimation result, standard deviation of the query estimation, and the confidence interval (CI) of the query estimation with a 95% confidence interval. The demo showed the ground truth query result is included within the CI as desired. This shows the system successfully provided the user with an accurate bounded answer including the ground truth query result.

The system displayed the elapsed time for bootstrap sampling, synopsis creation time, and the total execution time.In reality, synopses can be created before a query is submitted which will not affect the query estimation speed.

7. EXPERIMENT

In this section, we benchmark the accuracy of the developed system when answering analytic queries, especially join queries, on variate scales of data. The setup of experiments is presented in Section 0. The design of accuracy tests is presented in Section 7.2. The impact of system hyperparameters such as sampling fractions (f), bootstrap resample iterations (B), and data volumes (sizes) are evaluated in sections 7.3, 7.4, and 7.5.

```
fyu:src$cargo run -- -d tpch_100m.db -s 1 -b 2000 2>/dev/null
sample_fraction 1.0%
bootstrap iteration: 2000
database: "tpch_100m"
query: "select count(*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c
_custkey and c_acctbal < 10000 and c_acctbal > 5000"
Join Condition: [
    Where {
        left: "l_orderkey",
        right: "o_orderkey",
        operator: "=",
    },
    Where {
        left: "o_custkey",
        right: "c_custkey",
        operator: "=",
    },
]
Selection Conditions: [
    Where {
        left: "c_acctbal",
        right: "10000",
        operator: "<",
    },
    Where {
        left: "c_acctbal",
        right: "5000",
        operator: ">",
    },
]
s1_sample table created with sampled data.
s2_sample table created with joined data.
s3_sample table created with joined data.
s4_sample table created with joined data.
s5_sample table created with joined data.
Database Ground Truth: 268102
Sample Ground Truth: 261100
Bootstrap Time Taken: 0.79s
Standard Error: 3897.62
CI: [253460.66, 268739.34] (with 95% confidence level)
The database ground truth 268102 is within the confidence interval 👍
Execution time: 1.90s
```

FIGURE5: System Runtime.

7.1 Experiment Setup

The experiment was performed on a server equipped with an Intel(R) Core(TM) i7-10710U CPU which operates at a base frequency of 1.10GHz. The system is equipped with 21GB of RAM and runs the operating system of CentOS 7 with the Linux kernel version 3.10. The experiment scripts are developed using the Python 3 programming language which interacts with the systemimplemented in Rust.

The datasets used for experiments are generated using the TPC-H benchmark (TPC-H Benchmark, 2024) in different data volumes, including 100 MB, 1 GB, and 10 GB. Since the focus is to test the accuracy of the query estimation, all test datasets are stored on a centralized database system, namely SQLite. The test queries for accuracy tests are generated by the TPC-H benchmark with randomvalues in the query conditions. The queries are grouped by the number of joins which ranges from one to four joins, where each group includes 10 test queries. To obtain reliable running results, each test query is executed 10 times, and the average running result is calculated.

The statistic synopses were created using CS2 (F. Yu *et al.*, 2013), where sampling fractions employed include 0.1%, 0.5%, and 1.0% which demonstrated synopses using small, medium, and large sampling sizes. The iterationsof bootstrap resample used include B=200 and B=2000 to test their impact on the accuracy of error estimation.

7.2 Accuracy Tests

The accuracy tests aim to evaluate the error assessment functionality and the accuracy of the bounded answers produced by the developed AQP system. After a query is submitted to the

developed AQP system, it will generate multiple responsesto a user including the ground truth query result, estimated query result, error assessment, and a confidence interval (CI) of the query estimation. The desired CI shall include the ground truth query result, i.e. the ground truth query result shall be between the lower and upper bound of the CI, or the ground truth query result "hit" within the CI; otherwise, the CI "misses" the ground truth query result. In order to quantify the accuracy performance of error assessment and the developed AQP system, we introduce an easy-understanding measurement named *hit percentage* (or *hit ratio*). The hit percentage during the tests can be calculated as follows.

$$\text{hit percentage} = \frac{\text{times(total hits)}}{\text{times(total experiments)}} \times 100\% \quad (3)$$

The test queries are grouped by the total number of joins. The hit percentage is calculated for each group of test queries. We analyze the impact of hyperparameters, including the sampling faction, bootstrap resample iteration, and dataset volume, on the hit percentage. In general, it is observed that the average hit percentages are above 90% across all figures. The hit percentages observed in the experiments are considerably accurate even given small sample fractions and few bootstrap resample iterations.

7.3 Impact of Sampling Fractions

Figures 6, 7, and 8 depict the experiment results of average hit percentages for various sample fractions (or sample sizes, f) in test queries. The results are further grouped by number of joins in test queries and the bootstrap resample iterations (B) employed. The statistic synopsis employed in the AQP system is based on the CS2 scheme with sampling fractions set to 0.1%, 0.5%, and 1% in each of the test datasets. These simulate the scenarios of tiny, medium, and large samples. In each group, the test results are further divided according to B set to 200 and 2000. These simulate the scenarios of small and large bootstrap resample iterations.

The volumes of AQP synopses were observed to progressively increase given larger sample fractions. This was due to more sampled data being included in the synopses. A larger synopsis includes more samples from the original raw database and will improve the accuracy of AQP estimations. As observed from the experiment results, when setting the B value fixed, the hit percentage generally increases when the sample fraction increases.

Another perspective is the changing of join numbers. When join numbers increase, the AQP estimation accuracy tends to be lower as the query becomes more complex. Therefore, the hit percentage tends to progressively decrease when the join number increases.

7.4 Impact of Bootstrap Resample Iterations

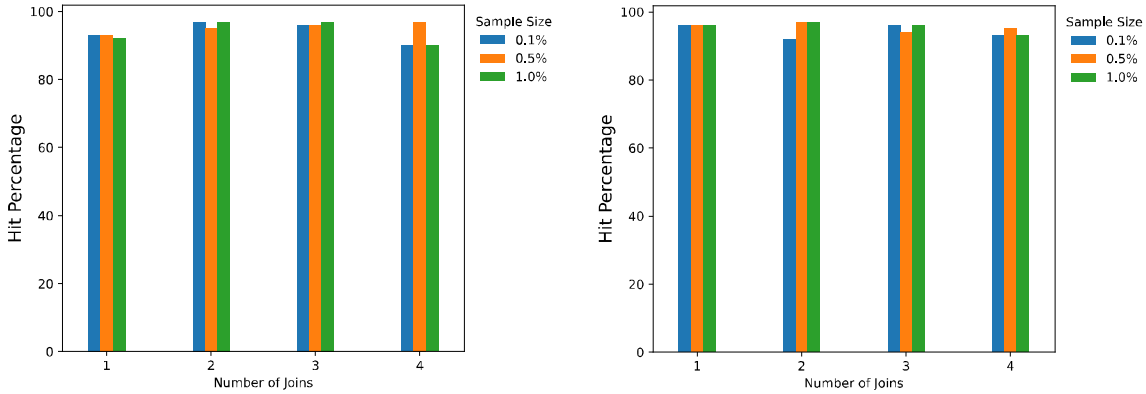
Figures 9, 10, and 11 depict the change of the average hit percentages given different numbers of bootstrap resample iterations in each group of join query and sample fraction. Take Figure6 (a) as an example, the sampling fraction is set to 0.1% and the join numbers in test queries are increasing from 1 to 4. The two hit percentages of bootstrap resample iterations, namely B=200 and B=2000, are depicted in each test query group respectively.

As observed in the results, the hit percentages of B=2000 are generally greater than B=200. Provided more iterations of bootstrap resampling, more bootstrap replications are produced which will increase the accuracy of error assessment. Therefore, setting the sampling fraction fixed, the hit percentage generally increases when more bootstrap resamples are computed.

7.5 Impact of Data Volumes

Figures 12, 13, and 14 depict the variation of hit percentage with fixed total bootstrap iterations and sampling fraction when data volume changes, including 100 MB, 1 GB, and 10 GB. The hit percentages are grouped according to the number of joins in test queries in the line graphs. As observed in the results, when B=200, there is no obvious impact of changing the data volumes.

However, when B=2000, the hit percentages gradually increase with the data volume. One reason can be, that when B=200, the error assessment accuracy may not be stable as there are fewer bootstrap resamples; however, when B=2000,the error assessment accuracy tends to be more stable. Another potential reason is the statistic synopsis tends to be more accurate when data volume is larger as more distinct tuples are sampled in a larger dataset even if the sample fraction is unchanged. In general, the AQP system produces better accuracy given larger datasets.



(a) 100 MB, B=200 (b) 100 MB, B=2000

FIGUREG: Hit percentage for 100 MB data set with different bootstrap iterations.

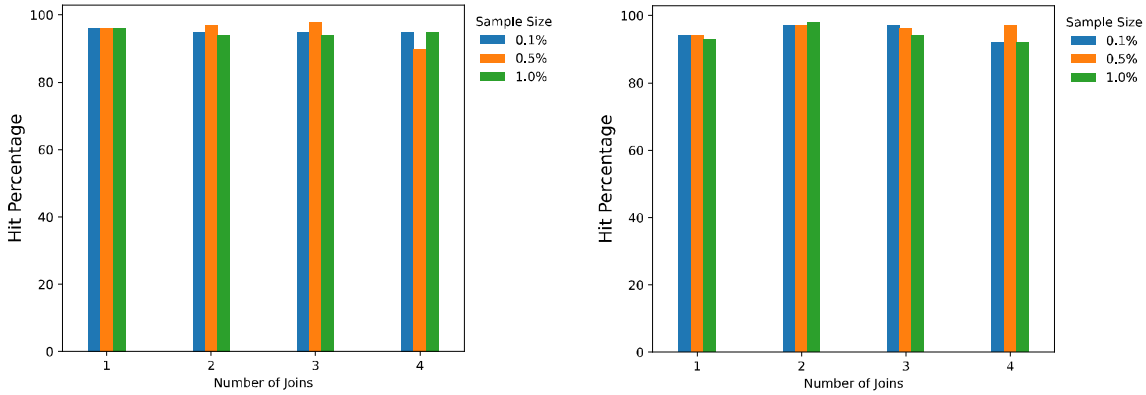
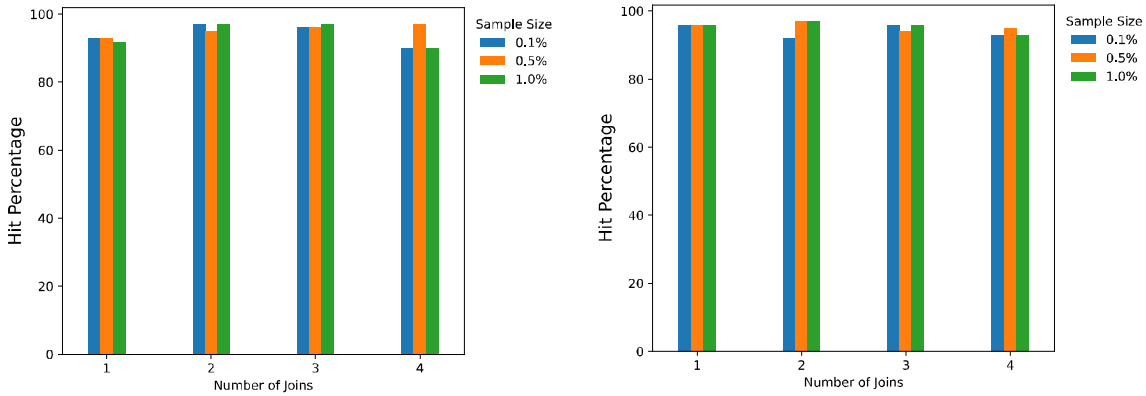
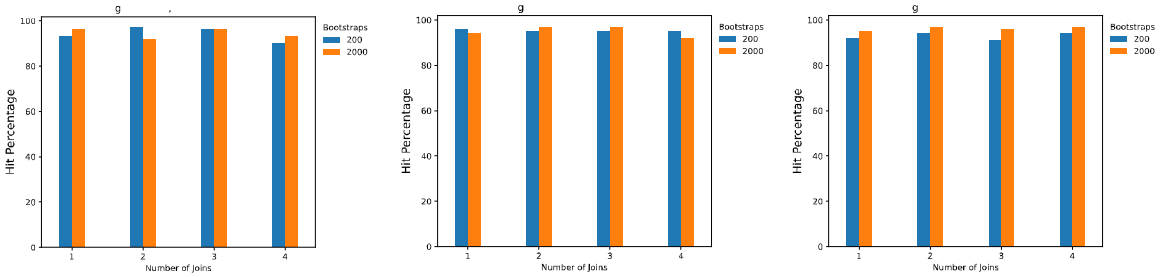


FIGURE7: Hit percentage for 1GB data set with different bootstrap iterations.



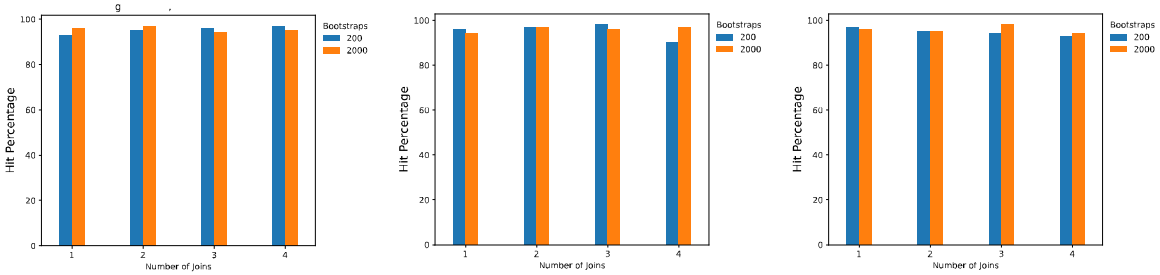
(a) 10 GB, B=200 (b) 10 GB, B=2000

FIGURE8: Hit percentage for 10GB data set with different bootstrap samples.



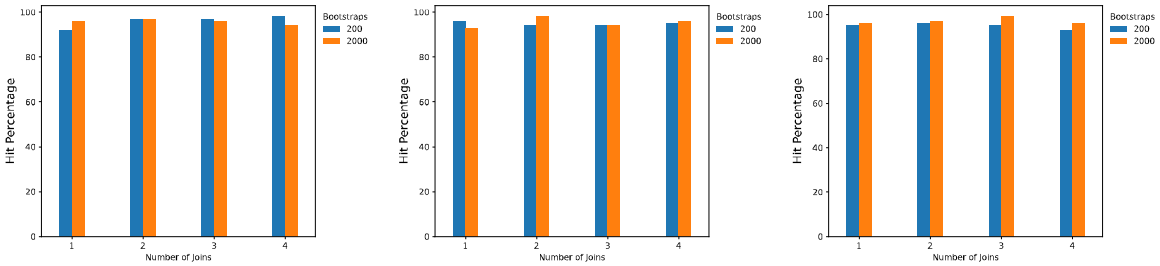
(a) 100 MB (b) 1 GB (a) 10 GB

FIGURE9: Hit percentage for 0.1% sample size and different data sizes.



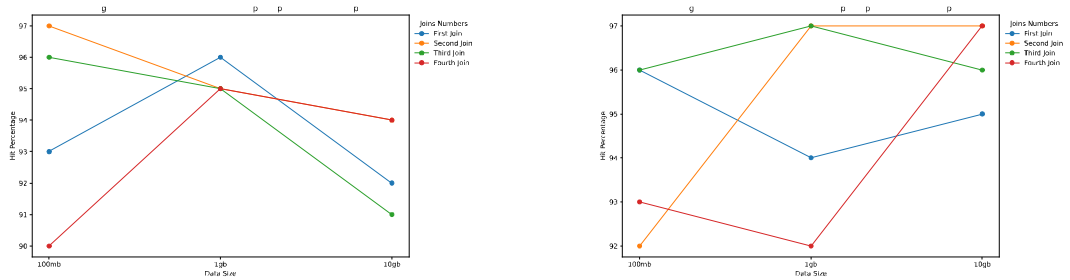
(a) 100 MB (b) 1 GB (a) 10 GB

FIGURE10: Hit percentage for 0.5% sample size and different data sizes.



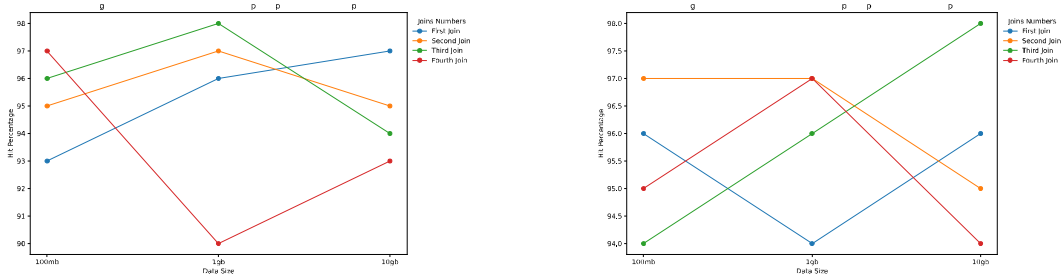
(a) 100 MB (b) 1 GB (a) 10 GB

FIGURE11: Hit percentage for 1% sample size and different data sizes.



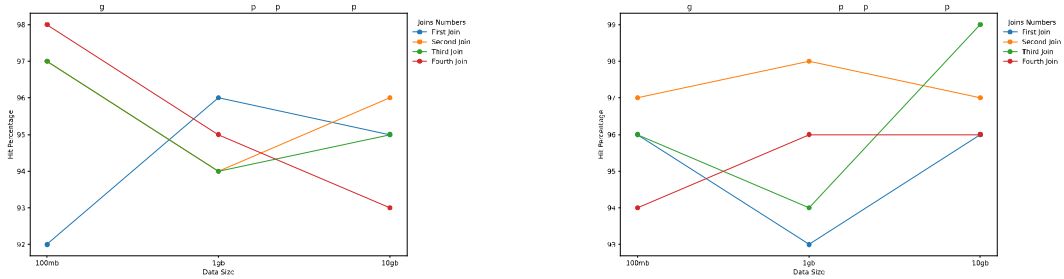
(a) B=200 (b) B=2000

FIGURE12: Line Graph for Different data sizes for sample fraction f=0.1%.



(a) B=200 (b) B=2000

FIGURE13: Line Graph for Different data sizes for sample fraction f=0.5%.



(a) B=200 (b) B=2000

FIGURE14: Line Graph for Different data sizes for sample fraction f=1%.

8. RELATED WORK

Existing work ofAQP broadly focuses on two different directions including the online AQP and offline AQP (K. Li *et al.*, 2018; S. Chaudhuri *et al.*, 2017).

The online AQP accepts a user query for estimation first and then collects statistic synopses for query estimation. These synopses can only be collected at query execution time and may require multiple iterations of sampling to achieve the desired accuracy requirement. New synopses must be collected when a different user query is submitted. In addition, most online AQP schemes heavily rely on high-end hardware and the existence of auxiliary data structures such as indices or hash tables on join attributes. Global indices and hash tables are not only costive to build but also expensive to maintain.

Examples of online AQP schemes include Wander Join (F. Li *et al.*, 2016), Two-Level Sampling (Y. Chen *et al.*, 2017), and Index-based Join Sampling (V. Leis *et al.*, 2017). Wander Join executes random walks to estimate the result of a join operator. A traditional random walk process randomly collects tuples from the joining tables to calculate the join result. Wander Join improves random walk to select only the tuples that are joinable in every step to retain the correlations in the synopsis. However, this procedure can be time-consuming given large datasets. To achieve a high sampling speed, it depends on the availability of indices on the join attributes. A later study, two-level sampling, improves the random walk process for join estimation by combining independent Bernoulli sampling, correlated sampling, and end-biased sampling. The two-level sampling can reach a higher accuracy for join estimations. The index-based join sampling is a join estimation technique focusing on querying in-memory data. It designs an algorithm that can utilize the join indices to quickly estimate the join size and choose the tuples to be sampled. This will avoid materializing any unnecessary join result and save computing costs and time. The index-based join sampling critically relies on the existence of join indices and focuses on the application to in-memory database systems.

Unlike online AQP, offline AQP schemes collect statistic synopses before the submission of user queries. This will avoid the need for high-end hardware or auxiliary data structure to hasten the sampling speed. The representative offline AQP schemes include Join Synopses (JS) (S.

Acharya *et al.*, 1999), Correlated Sampling Synopsis (CS2) (F. Yu *et al.*, 2013), and Tuple Graph Synopses (TuG) (J. Spiegel *et al.*, 2009).

JS employs correlated sampling to collect joinable tuples across multiple relations to keep the correlations for join query estimation. Once the JS is created, all future queries can be estimated, without the need to re-collect the JS, given the database join graph is unchanged. One constraint of JS is that it strongly requires simple random sampling without replacement (SRSWOR) on each of the relations in a database. For a database with a complicated join graph and long chains of joinable tables, building the JS can be considerably complicated. For example, for a join chain of N relations, the sampling cost is at least $O(N^2)$ which is expensive when N is large. Second, JS can only be applied for foreign-key join estimations. It cannot estimate many-to-many joins or cyclic join queries which are more complex. Third, JS cannot process dangling tuples in relations that are not joinable with tuples in other relations.

CS2 improves JS by not requiring SRSWOR on each relation in the join graph. Like JS, CS2 also developed join query estimators that can estimate general join queries. A special type of join query, named no-source query, does not include the table where a SRSWOR exists. For this type of query, CS2 introduced a special value named JR value along with the RV estimator to derive unbiased join query estimations.

TuG was inspired by utilizing a semi-structured view, such as XML, for a relational database. It summarizes a relational database using graph-model synopses to retain the joinable relationships between tuples as well as the selectivity information of tuple attribute values. TuG also introduced methods to reduce the volumes of the synopses to save storage costs. However, the space reduction of TuG will significantly lower the accuracy of query estimation. Another drawback of TuG is it requires a long time to generate synopses given a large dataset. Compared with the sample-based synopses, namely JS and CS2, TuG requires more time to build and produces less accurate estimation given the same synopsis storage constraint.

Multipleonline AQP systems have been developed including DBO (C. Jermaine *et al.*, 2008), BlinkDB (S. Agarwal *et al.*, 2013), VertictDB(Y. Park *et al.*, 2018), and XDB (F. Li *et al.*, 2019). Among them, BlinkDB can provide both query estimation and enable constraints of querying time limit and error bounds. It is a distributed online AQP system that can work on big data platforms such as Hadoop and Hive. BlinkDB is mainly designed to estimate select queries. VeridictDB introduced a database learning approach that uses previous query results to iteratively improve future query estimations. XDB can answer join queries based on the wander join framework. To reach the high sampling speed required by online AQP, the above systems either require high-speed hardware or expensive auxiliary data structures.

Unlike online AQP systems, AQPrius is based on the offline AQP scheme, namely CS2, which doesn't require a high sampling speed and can significantly reduce computing costs. AQPrius is a practical system implemented in Rust and fits well in situations where approximated query answers are needed in a short time such as an exploratory data analysis on big data. In addition, AQPrius doesn’t require high-end computing devices. Therefore, itcan be a suitable solution to answer queries on big data given limited computing resources such as in mobile and edge computing.

9. CONCLUSION

In this paper, we presented AQPrius, a comprehensive offline AQP system that can approximatecomplex queries on big data and provide error assessment for query estimations. The AQPrius employs offline AQP to avoid the requirements of high-end hardware or expensive auxiliary data structures. A unique feature of AQPrius is the employment of a non-parametric statistic technique, named bootstrap sampling, to estimate the standard error of query estimation. Bootstrap sampling can provide error assessments even when the population distributions are unknown. The standard error provided by the bootstrap sampling can also be used to provide bounded query estimations (or confidence intervals) to enhance user feedback. We evaluated

AQPrius by utilizing the well-known TPC-H benchmarks. The experimental results showed that AQPrius can precisely answer complex queries, such as join queries, with high efficiency. The confidence intervals produced by the system are accurate even the given settings of tiny sampling fractions and fewer bootstrap iterations.

In the future, we would like to implement AQPrius to answer more complex query types such as the GROUPBY queries. We will extend the AQPrius to answer more complicated join queries such as no-source join queries.A web interface will be developed to improve the system's usability.

10. ACKNOWLEDGEMENT

This work was partially supported by the Research Professorship at Youngstown State University.

11. REFERENCES

B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable approximate query processing with the DBO engine," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 1–54, 2008, doi: 10.1145/1412331.1412335.

D. L. Quoc *et al.*, "Approximate Distributed Joins in Apache Spark," *ArXiv e-prints*, vol. abs/1805.0, May 2018, [Online]. Available: http://arxiv.org/abs/1805.05874

D. Wilson, W.-C. Hou, and F. Yu, "Scalable Correlated Sampling for Join Query Estimations on Big Data," in *Proc. of 28th International Conference on Software Engineering and Data Engineering*, F. Harris, S. Dascalu, S. Sharma, and R. Wu, Eds., EasyChair, 2019, pp. 41–50. doi: 10.29007/87vt.

F. Li *et al.*, "Wander Join: Online Aggregation via Random Walks," *Proc. SIGMOD'16*, pp. 615–629, 2016.

F. Li, B. Wu, K. Yi, and Z. Zhao, "Wander Join and XDB: Online Aggregation via Random Walks," *ACM Trans. Database Syst.*, vol. 44, no. 1, p. 2:1-2:41, Jan. 2019.

F. Yu, W.-C. Hou, C. Luo, D. Che, and M. Zhu, "CS2: A New Database Synopsis for Query Estimation," in *Proc. SIGMOD'13*, ACM, 2013, pp. 469–480. doi: 10.1145/2463676.2463701.

F. Yu, W.-C. Hou, C. Luo, D. Che, and M. Zhu, "CS2: a new database synopsis for query estimation," in *SIGMOD 2013*, ACM, 2013, pp. 469–480.

J. Bater, Y. Park, X. He, X. Wang, and J. Rogers, "Saqe: practical privacy-preserving approximate query processing for data federations," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2691–2705, 2020.

J. Spiegel and N. Polyzotis, "TuG synopses for approximate query answering," *ACM Trans. Database Syst.*, vol. 34, no. 1, p. 3:1—-3:56, Apr. 2009, doi: 10.1145/1508857.1508860.

K. Li and G. Li, "Approximate query processing: what is new and where to go?," *Data Science and Engineering*, vol. 3, no. 4, pp. 379–397, 2018.

M. Sch, J. Schildgen, and S. Deßloch, "Sampling with Incremental MapReduce," in *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2015.

Q. Liu, "Approximate Query Processing," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds., Springer US, 2009, pp. 113–119. doi: 10.1007/978-0-387-39940-9_534.

S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, “Join Synopses for Approximate Query Answering,” in *Proc. SIGMOD’99*, ACM, 1999, pp. 275–286.

S. Agarwal *et al.*, “BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data,” in *Eurosys’13*, 2013, pp. 29–42. doi: 10.1145/2465351.2465355.

S. Agarwal *et al.*, “Knowing When You’re Wrong: Building Fast and Reliable Approximate Query Processing Systems,” in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data - SIGMOD*, 2014, pp. 481–492. doi: 10.1145/2588555.2593667.

S. Chaudhuri, B. Ding, and S. Kandula, “Approximate query processing: No silver bullet,” in *Proc. SIGMOD’17*, 2017, pp. 511–519.

“TPC-H Benchmark.” [Online]. Available: https://www.tpc.org/tpch/

T. Siddiqui, A. Jindal, S. Qiao, H. Patel, and W. Le, “Cost models for big data query processing: Learning, retrofitting, and our findings,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 99–113.

T. Tian, “Social big data: techniques and recent applications,” *International Journal of Computer Science and Security (IJCSS)*, vol. 14, no. 5, p. 224, 2020.

V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann, “Cardinality Estimation Done Right : Index-Based Join Sampling,” in *Proc. CIDR’17*, 2017.

Y. Chen and K. Yi, “Two-Level Sampling for Join Size Estimation,” in *Proc. ICDE’17*, ACM, 2017, pp. 759–774. doi: 10.1145/3035918.3035921.

Y. Park, B. Mozafari, J. Sorenson, and J. Wang, “VerdictDB: universalizing approximate query processing,” in *Proc. SIGMOD’18*, ACM, 2018, pp. 1461–1476.

Z. Zhou, H. Zhang, S. Li, and X. Du, “Hermes: A Privacy-Preserving Approximate Search Framework for Big Data,” *IEEE Access*, vol. 6, pp. 20009–20020, 2018, doi: 10.1109/ACCESS.2017.2788013.