**ABS: a System for Scalable Approximate Queries with Accuracy Guarantees**

Kai Zeng†, Shi Gao†, Jiaqi Gu†, Barzan Mozafari‡, Carlo Zaniolo†

†University of California, Los Angeles
t‡University of Michigan

{kzeng, gaoshi, gujiaqi, zaniolo}@cs.ucla.edu
tmozafari@umich.edu

**ABSTRACT**

Approximate Query Processing (AQP) based on sampling is critical for supporting timely and cost-effective analytics over big data. To be applied successfully, AQP must be accompanied by reliable estimates on the quality of sample-produced approximate answers; the two main techniques used in the past for this purpose are (i) closed-form analytic error estimation, and (ii) the bootstrap method. Approach (i) is extremely efficient but lacks generality, whereas (ii) is general but suffers from high computational overhead. Our recently introduced Analytical Bootstrap method combines the strengths of both approaches and provides the basis for our ABS system, which will be demonstrated at the conference. The ABS system models bootstrap by a probabilistic relational model, and extends relational algebra with operations on probabilistic relations to predict the distributions of the AQP results. Thus, ABS entails a very fast computation of bootstrap-based quality measures for a general class of SQL queries, which is several orders of magnitude faster than the standard simulation-based bootstrap. In this demo, we will demonstrate the generality, automaticity, and ease of use of the ABS system, and its superior performance over the traditional approaches described above.

**Categories and Subject Descriptors**

H.2.4 [Systems]: Query processing

**Keywords**

Approximate Query Processing; Error Estimation; Bootstrap

**1. INTRODUCTION**

Today’s business, science and engineering disciplines are predominantly data-driven. The ever-growing size of data calls for timely and cost-effective analysis. This situation has brought even more attention to the already-active area of Approximate Query Processing (AQP). As a critical and general approach for coping with massive datasets, sampling is widely used in databases [4, 6, 9, 11, 12, 13, 18], Map-Reduce systems [5, 16], and data stream management systems [7, 17].

Sampling refers to the commonly used technique of evaluating the queries from small random sample of the original database.

The quality of the obtained approximate query answers plays an important role to their utility. As an example, during exploratory analysis, the analyst will seek assurance that the answer derived from the small sample is of “good quality”, e.g., within ±1% of the correct answer with probability ≥ 95%. Thus, assessing the quality (i.e. error estimation) is a fundamental aspect of AQP.

The past two decades have seen much work on error estimation, which can be categorized into two main approaches. The first approach [5, 8, 9, 11, 19, 22] analytically derives closed-form error estimates. Although computationally appealing, analytic error quantification is restricted to a very limited set of queries (simple group-by-aggregate queries) [20].

To address this problem, a second approach, named bootstrap, has emerged as a more general method for error estimation [15, 16, 20], and is becoming increasingly popular due to its wide applicability and automaticity [15, 16, 20]. Bootstrap [10, 21] is a Monte-Carlo procedure, which given an initial sample (i) repeatedly forms simulated datasets by resampling tuples i.i.d. (identically and independently) from the given sample, (ii) recomputes the query on each of the simulated datasets, and (iii) assesses the quality of answer on the basis of the empirical distribution of the produced query answers. However, bootstrap is highly computation-demanding, since it requires hundreds or even thousands of trials to obtain a reliable estimate [15, 20].

In this demonstration, we introduce the ABS system, a fast error estimation system for AQP. The ABS system is designed and developed to bridge the gap between the two aforementioned approaches by dovetailing their merits while avoiding their limitations: ABS inherits the general and automatic nature of bootstrap, and thus can be applied to a much more general class of SQL queries, but does not require the Monte-Carlo simulation, and thus is highly computationally efficient. These merits of ABS enable complex exploratory data analysis on large volumes of data.

ABS achieves these merits by exploiting a new technique, called the analytical bootstrap. Analytical bootstrap succinctly models the set of all possible simulated datasets generated by bootstrap trials as a single probabilistic multiset database (PMDB for short), by annotating each tuple in the database with an integer-valued random variable. The random variable represents the possible multiplicity with which the tuple would appear in the simulated datasets. Then, ABS extends relational operators to manipulate these random variables during query evaluation, which produces an annotated relation where the annotations encode the distribution of all possible answers that could be generated by bootstrap. In particular, ABS evaluates the query only once, but can accurately estimate the empirical distribution of the query answers that would be produced by hundreds or thousands of bootstrap trials. Furthermore, analyti-
cral bootstrap can be easily integrated into existing database engines through user-defined types and user-defined functions.

During the demonstration, users will experience, through hands-on experience on real-life databases and queries, the convenience and efficiency of using ABS for error estimation in approximate query answering. The user interacts with ABS by simply providing the target query and specifying the desired error measure. The ABS will transparently rewrite the query to add quality quantification support, execute the rewritten query and deliver the query result as well as the quality measure in an interactive manner. The user can also compare analytical bootstrap against the closed-form approach and the standard simulation-based bootstrap to experience the generality and efficiency of the ABS system.

The rest of the paper is organized as follows: Section 2 provides a summary of the theoretical background. We present the system architecture in Section 3. Section 4 briefly describes the demonstration we are proposing. We conclude in Section 5.

2. ANALYTICAL BOOTSTRAP

We briefly review the background on bootstrap, and exemplify the analytical bootstrap method which is the key technique of the ABS system. Interested readers are referred to [23] for more details on analytical bootstrap.

Sampling and Bootstrap Sampling is widely used in approximate query processing, which consists in (i) taking a random sample \( D \) from the original database, (ii) evaluating a potentially modified query \( q(D) \) as an approximation for the original query. However, the obtained approximate results are of little use if they are not accompanied with accuracy estimation.

Bootstrap [10, 21] is powerful tool for estimating the quality of \( q(D) \), which consists in a simple Monte Carlo procedure: it repeatedly carries out a sub-routine, called a trial. Each trial generates a simulated database, say \( D_s \), which is of the same size as \( D \) (by sampling \( |D| \) tuples i.i.d. from \( D \) with replacement), and then computes query \( q(D_s) \) on \( D_s \). Consider a simplified version of the \textit{lineitem} relation and the “Small-quantity-order Revenue” query \( q \) (see Example 1) from the TPC-H benchmark [3]. A sample \( D \) of \textit{lineitem} is shown in Figure 1(a).

Figure 1(b) is an instantiation of Figure 3(f) is the PMDB modeling all possible bootstrap resamples of \( F(X) \), where the nondeterministic multiplicities \( (\pi_1, \pi_2, \pi_3, \pi_4, \pi_5) \) jointly follow a multinomial distribution \textit{Multinomial}(5, [0.2, 0.2, 0.2, 0.2, 0.2]). Figure 1(b) is an instantiation of Figure 3(f) by assigning \( \pi_1 = 1, \pi_2 = 2 \) and so on.

Extending Relational Algebra Evaluating queries on bootstrap resamples simply follows relational algebra with multiset semantics. Green et al. [14] showed that one can query a multiset database by extending the relational algebra with the \( + \) and \( \otimes \) operators, which manipulate the annotated multiplicities: for projection, we add the multiplicities of all input tuples that are projected to the same result tuple, while for join, we multiply the multiplicities of joined tuples.

I.e., inductively:

- **Selection** \( \sigma(c)(R) \). \( \pi_{\sigma(c)}(R)(t) = \pi_R(t) \cdot I(c(t)) \), where \( I(c(t)) \) returns 1 if \( c(t) \) is true and 0 otherwise.
- **Projection** \( \Pi_A(R) \). \( \pi_{\Pi_A(R)}(R)(t) = \sum_{t[A]=t'} \pi_R(t') \) where \( t'[A] \) is the projection of \( t' \) on \( A \).
- **Join** \( R_1 \odot R_2 \). \( \pi_{\Pi_A(R)}(R)(t) = \pi_{R_1}(t_1) \cdot \pi_{R_2}(t_2) \), where \( t_1 \) is \( t \) on \( U_1 \).

Consider the query plan for Example 1 as shown in Figure 2.

Figure 2: Query plan for the running example

Even with the extended relational algebra, the simulation-based bootstrap still needs to evaluate the query many times on different resample instances. In contrast, the analytical bootstrap extends relational algebra with operators that directly manipulate the annotations in the PMDB, i.e., random variables representing the non-deterministic multiplicities. Specifically, the analytical bootstrap introduces two operators on the annotated random variables, namely \( + \) convolution (\( \oplus \)) and \( \odot \) convolution (\( \otimes \)), defined as: for any two random variables \( r_1 \) and \( r_2 \), \( r_1 \odot r_2 (r_1 \odot r_2) \) is a new random variable, where

\[
\Pr(r_1 \otimes r_2 = s) = \sum \{ \Pr(r_1 = x) \cdot \Pr(r_2 = y) \mid \forall x, y, x+y = s \}
\]

\[
\Pr(r_1 \odot r_2 = s) = \sum \{ \Pr(r_1 = x) \cdot \Pr(r_2 = y) \mid \forall x, y, x \cdot y = s \}
\]

\( \gamma_{AB}(B) \) denotes applying aggregate \( \alpha \) on \( B \) group by \( A \).
Figure 3: (a) One resample instance with annotations, (b)-(e) evaluation steps in multiset semantics, (f) the PMDB for Figure 1(a), and (g)-(j) evaluation steps of analytical bootstrap.

Figure 4: ABS Architecture

Similar to the extended relational algebra for multiset semantics, analytical bootstrap applies $\oplus$ operator whenever summing the annotations, and applies $\otimes$ operator whenever multiplying the annotations. Figure 3(g) to 3(j) show the corresponding evaluation steps of analytical bootstrap for Figure 2.

Efficient Evaluation It is both space and time consuming to manipulate random variables symbolically. Thus, at query time, analytical bootstrap represents and manipulates the random variables by their marginal distributions. Specifically, analytical bootstrap represents each annotation $\pi$ by a pair $(n, p)$, namely the multinomial representation, where $(i)$ $n$ is the number of multinomial trials in a bootstrap trial, which is the size of the relation being resampled, and $(ii)$ $p$ is the probability vector of a single multinomial trial, i.e., probability of this tuple being picked ($p[1]$) or not picked ($p[0]$). E.g., $\pi_1$ is represented by $(5, [p[1] = 0.2, p[0] = 0.8])$.

In contrast to symbolic manipulation, this evaluation technique can only be applied to the cases where tuples in any intermediate results are generated from disjoint set of tuples from the base relation. Fortunately, query plans that satisfy the requirement, called eligible plans [23], can be detected at compile time.

3. SYSTEM ARCHITECTURE

Figure 4 shows the high-level architecture of the ABS system, which can be divided into two main components: (1) Query Translation Engine: transparently compiling, checking and rewriting the query to support error estimation. (2) DB Execution Engine: evaluating the query augmented with error estimation operations, and delivering the accuracy measures in user-specified metrics. The two components are decoupled by the rewritten query plans:

- **Query Translation Engine** Taking a SQL query, the compiler generates a query plan expressed in relational algebra. Then, the compiler passes the execution plan along with the basic settings of the input database to the eligibility checking module. Based on the eligibility rules defined in [11, 23], the eligibility checking module verifies if the input query plan is suitable to perform analytical bootstrap and/or the closed-form method.

If an eligible plan is found, the compiler passes the plan to the rewriter. The rewriter takes into consideration the user-specified quality measures, and rewrites the plan into a new query plan with annotation enhanced operations in order to support error estimation, i.e., with additional annotations, and functions that propagate the annotations according to the methods discussed in Section 2. The rewritten query plan preserves the result of the original query, and adds the desired error measures specified by the user.

- **DB Execution Engine** The rewritten query plan is then submitted to the execution engine. Through user-defined type and user-defined aggregates/functions, the execution techniques of the ABS system, i.e., extended relational algebra in Section 2, can be easily integrated into common database engines as an extension module (ABS extension as shown in Figure 4). Specifically, the ABS system expresses the annotations in the form of user-defined types, and the convolution operations of the annotations as user-defined aggregates/functions. Currently, we implement the ABS system on top of Hive [1], an open source distributed data warehouse that supports efficient query evaluation on massive data sets. Furthermore, since our implementation is built as an extension module of Hive, it is easy to deploy ABS on other query engines (e.g., Shark [2]).

4. DEMONSTRATION DESCRIPTION

The demonstration is organized into three phases: (1) a brief introduction to the main system functionalities, in which we will exhibit the key components and features of our system; (2) a “hands-on” phase in which the public is invited to directly interact with the system and test its capabilities; and (3) a performance comparison, in which we demonstrate the superior performance of ABS by comparing it against the closed-form approach and the standard simulation-based bootstrap.

In the demonstration of the system functionalities, we will show how the system interface (e.g., figure 5) guides the user through the steps of query processing:

- (i) **Query Eligibility Verification**. In this step, the ABS system first loads the database and the corresponding database sample speci-
The demonstration highlights the main functionalities of the system, and exhibits how this important error estimation technique can be easily integrated into existing database systems.

6. REFERENCES