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 [‡]University of Michigan
[†]{kzeng, gaoshi, gujiaqi, zaniolo}@cs.ucla.edu [‡]mozafari@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 froma 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 $\pm 1\%$ of the correct answer with probability $\geq 95\%$. 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 boostrap 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-

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594532.

cal 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 on D, and (iii) using $q(D)$ 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 $\hat{D}_i$, which is of the same size as D (by sampling $|D|$ tuples i.i.d. from D *with* replacement), and then computes query q on $\hat{D}_i$. Consider a simplified version of the *lineitem* relation and the "Small-quantity-order Revenue" query q (see Example 1) from the TPC-H benchmark [3]. A sample D of *lineitem* is shown in shown in Figure 1(a).[1] Figure 1(b) and 1(c) show one possible resample from a bootstrap trial and the corresponding query result, respectively. The collection $\{q(\hat{D}_i)\}$ from all the bootstrap trials forms an empirical distribution, based on which various accuracy measures, e.g., variance and confidence intervals, can be computed.

EXAMPLE 1 (SMALL-QUANTITY-ORDER REVENUE).
```
SELECT l_partkey, SUM(l_extendedprice) as revenue
FROM    lineitem as outer
WHERE   l_quantity < (
        SELECT  SUM(l_quantity) / 3
        FROM    lineitem as inner
        WHERE   inner.l_partkey = outer.l_partkey)
GROUP BY l_partkey
```

Bootstrap is effective and robust over a wide range of practical situations [15, 20, 16]. However, as mentioned above, bootstrap suffers from high computational overhead, as it requires hundreds or even thousands of bootstrap trials in order to obtain an accurate quality estimate. Next, we introduce analytical bootstrap, which has been proven equivalent to the simulation-based bootstrap, but avoids the computational overhead [23].

Probabilistic Multiset Database Each bootstrap resample generates a multiset relation. E.g., in the resample shown in Figure 1(b),

[1]We use *l_pk*, *l_q* and *l_ep* as short for *l_partkey*, *l_quantity* and *l_extendedprice*, respectively.

lineitem

$D =$

	l_pk	l_q	l_ep
t_1	p01	4	20
t_2	p01	5	25
t_3	p01	3	15
t_4	p02	1	10
t_5	p02	8	80

(a)

lineitem

$\hat{D} =$

tuple
t_1
t_2
t_2
t_4
t_5

(b)

l_pk	revenue
p01	20
p02	10

(c)

Figure 1: (a) An example database sample D, (b) a resample of D, and (c) the corresponding query result

tuple t_2 is drawn twice, while tuple t_3 is not selected. These multiset relations can be represented in a functional way, where each tuple t is annotated with an integer $\pi(t)$, representing its multiplicity in the resample. Figure 3(a) shows the functional representation of Figure 1(b).

The key idea of analytical bootstrap is to model all possible bootstrap resamples as a single relation, where tuples are annotated with nondeterministic multiplicities, and thus a query can be evaluated once on this relation, instead of on many different resamples. Since the annotations are probabilistic, the obtained database is called a *probabilistic multiset database* (PMDB). For instance, Figure 3(f) is the PMDB modeling all possible bootstrap resamples of Figure 1(a), where the nondeterministic multiplicities $(\pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$ jointly follow a multinomial distribution $Multinomial(5, [0.2, 0.2, 0.2, 0.2, 0.2])$. 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 $\cdot$ 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** $\sigma_c(R)$. $\pi_{\sigma_c(R)}(t) = \pi_R(t) \cdot \mathbb{1}(c(t))$, where $\mathbb{1}(c(t))$ returns 1 if $c(t)$ is true and 0 otherwise.
- **Projection** $\Pi_A(R)$. $\pi_{\Pi_A(R)}(t) = \sum_{t'[A]=t} \pi_R(t')$ where $t'[A]$ is the projection of t' on A.
- **Join** $R_1 \bowtie R_2$. $\pi_{R_1 \bowtie R_2}(t) = \pi_{R_1}(t_1) \cdot \pi_{R_2}(t_2)$, where t_i is t on U_i.

Consider the query plan for Example 1 as shown in Figure 2.[2] Figure 3(b) to 3(e) demonstrate the evaluation steps using the extended relational algebra.

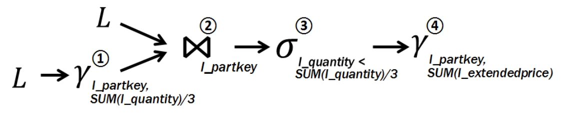

L
L
①
l_partkey, SUM(l_quantity)/3
②
l_partkey
③
l_quantity < SUM(l_quantity)/3
④
l_partkey, SUM(l_extendedprice)

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 nondeterministic multiplicities. Specifically, the analytical bootstrap introduces two operators on the annotated random variables, namely $+$ *convolution* ($\oplus$) and $\cdot$ *convolution* ($\odot$), defined as: for any two random variables r_1 and r_2, $r_1 \oplus r_2$ ($r_1 \odot r_2$) is a new random variable, where

$$\Pr(r_1 \oplus r_2 = s) = \sum\{\Pr(r_1 = x \wedge r_2 = y) \mid \forall x, y, x + y = s\}$$
$$\Pr(r_1 \odot r_2 = s) = \sum\{\Pr(r_1 = x \wedge r_2 = y) \mid \forall x, y, x \cdot y = s\}$$

[2]$\gamma_{A,\alpha(B)}$ denotes applying aggregate α on B group by A.

lineitem

t	π
t_1	1
t_2	2
t_3	0
t_4	1
t_5	1

(a)

l_pk	$\varpi_{\textbf{sum}}$	π
p01	$4 \times 1 + 5 \times 2 + 3 \times 0 = 14$	$\mathbb{1}(1 + 2 + 0) = 1$
p02	$1 \times 1 + 8 \times 1 = 9$	$\mathbb{1}(1 + 1) = 1$

(b) Step ① in Figure 2

t	$\varpi_{\textbf{sum}}$	π
t_1	14	$1 \times 1 = 1$
t_2	14	$2 \times 1 = 2$
t_3	14	$0 \times 1 = 0$
t_4	9	$1 \times 1 = 1$
t_5	9	$1 \times 1 = 1$

(c) Step ② in Figure 2

t	π
t_1	$\mathbb{1}(14 > 12) \times 1 = 1$
t_2	$\mathbb{1}(14 > 15) \times 1 = 0$
t_3	$\mathbb{1}(14 > 9) \times 0 = 0$
t_4	$\mathbb{1}(9 > 3) \times 1 = 1$
t_5	$\mathbb{1}(9 > 24) \times 1 = 1$

(d) Step ③ in Figure 2

l_pk	$\varpi_{\textbf{SUM}}$
p01	$1 \times 20 + 0 \times 25 + 0 \times 15 = 20$
p02	$10 \times 1 + 80 \times 0 = 10$

(e) Step ④ in Figure 2

lineitem

t	π
t_1	π_1
t_2	π_2
t_3	π_3
t_4	π_4
t_5	π_5

(f)

l_pk	$\varpi_{\textbf{sum}}$	π
p01	$\varpi_1 = 4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3$	$\pi'_1 = \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$
p02	$\varpi_2 = 1\pi_4 \oplus 8\pi_5$	$\pi'_2 = \mathbb{1}(\pi_4 \oplus \pi_5)$

(g) Step ① in Figure 2

t	$\varpi_{\textbf{sum}}$	π
t_1	ϖ_1	$\pi_1 \odot \pi'_1$
t_2	ϖ_1	$\pi_2 \odot \pi'_1$
t_3	ϖ_1	$\pi_3 \odot \pi'_1$
t_4	ϖ_2	$\pi_4 \odot \pi'_2$
t_5	ϖ_2	$\pi_5 \odot \pi'_2$

(h) Step ② in Figure 2

t	π
t_1	$\pi''_1 = \mathbb{1}(\varpi_1 > 12) \odot \pi_1 \odot \pi'_1$
t_2	$\pi''_2 = \mathbb{1}(\varpi_1 > 15) \odot \pi_2 \odot \pi'_1$
t_3	$\pi''_3 = \mathbb{1}(\varpi_1 > 9) \odot \pi_3 \odot \pi'_1$
t_4	$\pi''_4 = \mathbb{1}(\varpi_2 > 3) \odot \pi_4 \odot \pi'_2$
t_5	$\pi''_5 = \mathbb{1}(\varpi_2 > 24) \odot \pi_5 \odot \pi'_2$

(i) Step ③ in Figure 2

l_pk	$\varpi_{\textbf{sum}}$
p01	$20\pi''_1 \oplus 25\pi''_2 \oplus 15\pi''_3$
p02	$10\pi''_4 \oplus 80\pi''_5$

(j) Step ④ in Figure 2

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

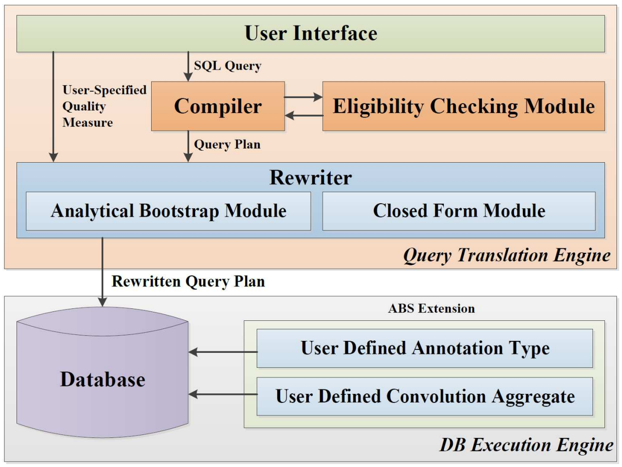


Figure 4: ***ABS*** **Architecture**

Similar to the extended relational algebra for multiset semantics, analytical bootstrap applies $\oplus$ operator whenever summing the annotations, and applies $\odot$ 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 π by a pair $(n, \mathbf{p})$, 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) $\mathbf{p}$ is the probability vector of a single multinomial trial, i.e., probability of this tuple being picked ($\mathbf{p}[1]$) or not picked ($\mathbf{p}[0]$). E.g., π_1 is represented by $(5, [\mathbf{p}[1] = 0.2, \mathbf{p}[0] = 0.8])$.

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.

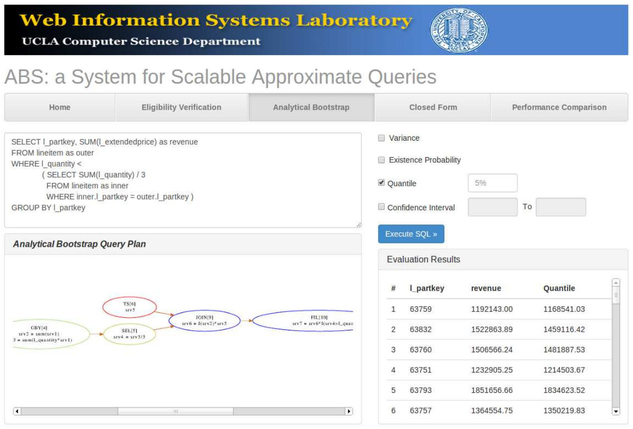
DB Execution Engine The rewritten query plan is then submitted to the execution engine. Through user-defined type and user-defined aggregates/functions, the execution techniques of the *ABS* system, i.e., extended relational algebra in Section 2, can be easily integrated into common database engines 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-


Web Information Systems Laboratory
UCLA Computer Science Department
ABS: a System for Scalable Approximate Queries
Home
Eligibility Verification
Analytical Bootstrap
Closed Form
Performance Comparison
Analytical Bootstrap Query Plan
TS[6] srv5
JOIN[9] srv6 = I(srv2)*srv5
FIL[10] srv7 = srv6*I(srv4>l_qua
GBY[4] srv2 = sum(srv1) 3 = sum(l_quantity*srv1)
SEL[5] srv4 = srv3/3
Variance
Existence Probability
Quantile
Confidence Interval
To
Execute SQL »
Evaluation Results

```
SELECT l_partkey, SUM(l_extendedprice) as revenue
FROM lineitem as outer
WHERE l_quantity <
          ( SELECT SUM(l_quantity) / 3
            FROM lineitem as inner
            WHERE inner.l_partkey = outer.l_partkey )
GROUP BY l_partkey
```

☐ Variance
☐ Existence Probability
☑ Quantile 5%
☐ Confidence Interval

#	l_partkey	revenue	Quantile
1	63759	1192143.00	1168541.03
2	63832	1522863.89	1459116.42
3	63760	1506566.24	1481887.53
4	63751	1232905.25	1214503.67
5	63793	1851656.66	1834623.52
6	63757	1364554.75	1350219.83

Figure 5: *ABS* Web Interface: analytical bootstrap evaluation

fied by user. Then the user can submit the query of interest to be evaluated on the loaded database. The system compiles the input query to a query plan explained in relational algebra, and provides users a visualization of the query plan. Based on this query plan, the system checks whether it is eligible for applying efficient analytical bootstrap method and/or closed-form method to estimate the approximation error of the query. Non-eligible operators will be marked out with detailed information indicating which eligible rule is violated.

In addition, the system will also provide a list of applicable error estimation methods, which could include one or more of (i) closed-form evaluation, (ii) analytical bootstrap method, and (iii) the standard bootstrap. If neither analytical bootstrap nor closed form evaluation is not applicable, the system will choose the standard bootstrap as default. At this point, the user can determine the evaluation method for the submitted query.
(ii) *Query Rewriting*. In this step, the *ABS* system presents how a query is rewritten into a new query that can provide error measures alongside the approximate answers. For analytical bootstrap evaluation, *ABS* extends the query plan by augmenting the query plan with annotation manipulation operations; while for closed form evaluation, the query plan will be augmented with functions to collect the statistics required in the prediction formulas, such as mean and variance of target columns in the query. The new query plan is then passed to our database engine.
(iii) *Query Evaluation and Error Measurement*. Once the rewritten query plan is ready, our database engine evaluates it on the chosen database sample. To demonstrate the result of query evaluation, the *ABS* system provides different kinds of error measures (e.g., variance, quantiles and confidence intervals) for the user to choose from based on the application purpose. Figure 5 shows the web interface of analytical bootstrap in *ABS*.
(iv) *Performance Comparison*. To better explore the various error estimation methods supported, the user can compare the methods in terms of the result accuracy and the evaluation efficiency. We have prepared several TPC-H datasets in different scales for comparison purposes. Since the standard bootstrap requires hundreds of iterations to reach reliable estimation, we have also prepared the bootstrap results for a few queries in advance.

5. CONCLUSION

The *ABS* system represents a major step forward for error estimation in AQP, achieving a level of generality, automaticity and superior efficiency that have not been obtained by previous approaches. 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

[1] Apache Hive Project. https://hive.apache.org/.
[2] Shark Project. http://shark.cs.berkeley.edu/.
[3] TPC-H Benchmark. http://www.tpc.org/tpch/.
[4] S. Acharya, P. B. Gibbons, et al. The Aqua Approximate Query Answering System. In *SIGMOD*, pages 574–576, 1999.
[5] S. Agarwal, B. Mozafari, et al. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*, pages 29–42, 2013.
[6] B. Babcock, S. Chaudhuri, et al. Dynamic Sample Selection for Approximate Query Processing. In *SIGMOD*, pages 539–550, 2003.
[7] B. Babcock, M. Datar, et al. Load Shedding for Aggregation Queries over Data Streams. In *ICDE*, page 350, 2004.
[8] M. Charikar, S. Chaudhuri, et al. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, pages 268–279, 2000.
[9] S. Chaudhuri, G. Das, et al. Optimized stratified sampling for approximate query processing. *TODS*, 32(2):9, 2007.
[10] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.
[11] J. M. Hellerstein, P. J. Haas, et al. Online Aggregation. In *SIGMOD*, pages 171–182, 1997.
[12] Y. Hu, S. Sundara, et al. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, pages 1104–1107, 2009.
[13] C. Jermaine, S. Arumugam, et al. Scalable Approximate Query Processing with DBO Engine. In *SIGMOD*, pages 1–54, 2007.
[14] G. Karvounarakis and T. J. Green. Semiring-Annotated Data: Queries and Provenance? *SIGMOD Record*, 41(3):5–14, 2012.
[15] A. Kleiner, A. Talwalkar, et al. A General Bootstrap Performance Diagnostic. In *KDD*, pages 419–427, 2013.
[16] N. Laptev, K. Zeng, et al. Early Accurate Results for Advanced Analytics on MapReduce. *PVLDB*, 5(10):1028–1039, 2012.
[17] B. Mozafari and C. Zaniolo. Optimal Load Shedding with Aggregates and Mining Queries. In *ICDE*, pages 76–88, 2010.
[18] C. Olston, E. Bortnikov, et al. Interactive Analysis of Web-Scale Data. In *CIDR*, 2009.
[19] N. Pansare, V. R. Borkar, et al. Online Aggregation for Large MapReduce Jobs. *PVLDB*, 4(11):1135–1145, 2011.
[20] A. Pol and C. Jermaine. Relational Confidence Bounds Are Easy With The Bootstrap. In *SIGMOD*, pages 587–598, 2005.
[21] A. van der Vaart and J. Wellner. *Weak Convergence and Empirical Processes*. Springer, corrected edition, Nov. 2000.
[22] S. Wu, B. C. Ooi, et al. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.
[23] K. Zeng, S. Gao, et al. The Analytical Bootstrap: a New Method for Fast Error Estimation in Approximate Query Processing. In *SIGMOD*, 2014.