Approximate Query Processing: No Silver Bullet

Surajit Chaudhuri, Bolin Ding, Srikanth Kandula
Microsoft Reseach
{surajitc,bolind,srikanth}@microsoft.com

ABSTRACT

In this paper, we reflect on the state of the art of Approximate Query Processing. Although much technical progress has been made in this area of research, we are yet to see its impact on products and services. We discuss two promising avenues to pursue towards integrating Approximate Query Processing into data platforms.

1. INTRODUCTION

While Big Data opens the possibility of gaining unprecedented insights, it comes at the price of increased need for computational resources (or risk of higher latency) for answering queries over voluminous data. The ability to provide approximate answers to queries at a fraction of the cost of executing the query in the traditional way, has the disruptive potential of allowing us to explore large datasets efficiently. Specifically, such techniques could prove effective in helping data scientists identify the subset of data that needs further drill-down, discovering trends, and enabling fast visualization. In this article, we will focus on approximate query processing schemes that are based on sampling techniques.

An approximate query processing (AQP) scheme can be characterized by the generality of the query language it supports, its error model and accuracy guarantee, the amount of work saved at runtime, and the amount of additional resources it requires in precomputation. These dimensions of an AQP scheme are not independent and much of the past work makes specific choices along the above four dimensions. Specifically, it seems impossible to have an AQP system that supports the richness of SQL with significant saving of work while providing an accuracy guarantee that is acceptable to a broad set of application workloads. Put another way, you cannot have it all.

If indeed there is no silver bullet, it begs the question if AQP is an impossible dream. In fact, even after decades of research, AQP remains largely confined to academic research and is not a well-established paradigm in today's products and services. Since applications routinely leverage application-specific approximations, the lack of adoption of AQP should not be viewed as evidence that approximation is uninteresting for applications. Instead, if we were to look at the current state of AQP, we will be led to believe that, while approximation is interesting for applications, approximation at the query processing layer has proven ineffective for applications, because either the semantics of error models and the accuracy guarantees of AQP or the extent of savings in work accrued by AQP have been unsatisfactory.

We firmly believe that the value proposition of AQP, outlined in the opening of this article, is considerable in the world of big data. We should not give up the pursuit of such systems. However, critical rethinking of our approach to AQP research is warranted with the exclusive goal of making such systems practical [30]. The first step in such rethinking is to be clear about what combinations of the four dimensions of AQP will make it possible for applications to find AQP systems attractive. In this article, we suggest two research directions to pursue based on our reflection.

One promising approach may be to cede control over accuracy to the user. This approach is based on accepting the reality that AQP systems will not be able to offer a priori (*i.e.*, before the query is executed by the AQP system) accuracy guarantee for arbitrary SQL queries. In such cases, it is better instead to equip the application programmer with language primitives that assist them with injecting approximation into their data intensive computation. To that end, we extend our query languages with sampling operators that enable the application programmer to express their application-specific semantics of accuracy. We will call such methods *query-time sampling* and discuss them in Section 2. To make sure that the programmer has sufficiently powerful options to use sampling in arbitrary SQL queries, novel samplers are needed. Like selection, sampling a relation can make all subsequent computation cheaper due to data reduction. However, unlike selection, pushing down sampling operators is not always possible. Thus, the introduction of new samplers introduces the novel cost-based query optimization challenge of identifying the most performant plan among plans that are equivalent while preserving the semantics of samplers. The performance gains depend on how deep in the query plan the samplers can be pushed and on the availability of physical access methods. Note that this model gives the programmer the ability to use the full power of relational languages (SQL and its variants) for their applications.

A second direction that we find attractive is one where AQP systems promise any incoming query an accuracy guarantee that is query-independent. Such a contract allows a uniform way to leverage the AQP system for exploratory queries instead of having to worry about potentially different amounts of error for each query. At this point, we can provide this guarantee only for OLAP class of queries (and not for all of SQL) and for a few error metrics. Moreover, to achieve significant savings in work at runtime, such an approach requires pre-computation. We discuss these methods in Section 3. The key intuitions are to leverage indices in a

SIGMOD'17 May 14-19, 2017, Chicago, IL, USA

ISBN 978-1-4503-4197-4/17/05.

DOI: http://dx.doi.org/10.1145/3035918.3056097

clever manner (*e.g.*, only for small groups) in addition to using precomputed samples. Unlike several prior techniques where accuracy estimates are computed a posteriori (*e.g.*, confidence intervals over samples), we will show how such approaches can provide a priori error bounds.

There is such a large body of work on AQP that we do not attempt to summarize all of it. Hence, we discuss all work related to the above two directions outlined above, and review some open problems in Section 4. Beyond technical advances in the above two directions (as well as any other approach that provides a clear value proposition to applications), more will be needed to make AQP a reality. To promote user adoption, we believe that it is crucially important to implement AQP in existing data platforms. It is also important to think about new scenarios that go beyond decision support queries and ask ourselves how approximate query processing may be of value. These two issues are briefly discussed in Section 5.

2. QUERY-TIME SAMPLING

In query-time sampling, the user explicitly specifies sampler operations in the query. The syntax from the SQL:2008 standard lets a user choose the sampler type, the sampling fraction and optionally other sampler parameters. A query optimizer can consider samplers as logical operators in the plan and can choose appropriate physical implementations (*e.g.*, reservoir sampling, sampling an index) in a cost-based manner. Each sampler operator has precise semantics and the query output accurately reflects the user specification. There is a rich history of work on query-time sampling beginning with [71, 75]. The key advantage of such sampling is that operations that follow the sampler can execute more quickly or with fewer resources. However, note that in this approach, the query execution layer only knows how to faithfully execute sampler operators. For general SQL queries, the querying system provides no statistical accuracy guarantee.[1] Early work [71] focused on a sampler operator that picks input rows uniformly-at-random with the given probability. Many databases and big data systems support such a uniform sample operator [2, 3, 4, 16, 25, 38, 66, 73, 80]. Online aggregation methods [28, 37, 50, 53, 57, 74] also uniformly sample the inputs. However, they do so in a progressive manner which leads to further challenges which we will discuss in §2.3. We begin by discussing some serious limitations of the uniform sampler operator which prevent it from being used universally, and introduce new sampler operators.

2.1 Beyond Uniform Sampling

It is natural for a user to want to insert samplers deep in their query (*e.g.*, sampling before a join or a selection) because doing so would save more work at runtime.

Groups and predicates: Consider executing the following query `SELECT ZipCode, AVG(Salary) FROM` $\mathcal{R}$ `GROUP BY ZipCode` on a uniform sample of the relation $\mathcal{R}$.

The uniform sample may miss the small groups entirely or yield too few rows leading to inaccurate estimates of `AVG` for small groups. Similar concerns arise when executing queries with predicates, *e.g.*, `SELECT COUNT(`$*$`) FROM` $\mathcal{R}$ `WHERE County` $=$ '`SanJuan`$'$, on a uniform sample of the input; the uniform sample may yield zero rows that pass the predicate or may yield too few rows leading to an inaccurate answer.

A standard solution for the above issue is to bias the sampling toward small groups and rows that pass predicates [10, 11, 13, 32]. An operator formulation for such sampling is the *distinct sampler*: given parameters $\mathcal{C}, f, p$, the distinct sampler yields at least f rows (if they exist) for each distinct value of column set $\mathcal{C}$ and all rows are passed with at least a probability p. A distinct sampler over the group-by or predicate columns will pass at least f rows for each value of these columns.

Because the distinct sampler executes at runtime in a query pipeline, unlike the case of general stratified sampling [10, 11, 13, 32], the distinct sampler has some stringent implementation requirements. Desirable properties for any sampler operator include finishing in one pass over data, having a small memory footprint and requiring no coordination at runtime among tasks that operate in parallel on partitions of data. A recent work shows how to achieve these properties for the distinct sampler [60].

Joins: In typical databases and certainly in big data systems, the data is spread across multiple tables. Since joins are costly, it is useful to ask whether sample-then-join can have equivalent accuracy to join-then-sample [34, 38]? If this is possible, then joins can execute over sampled inputs.

Most methods only support the special case of many-to-one equijoins; *e.g.*, key foreign-key or multiple such joins in a star schema [11, 13, 64]. In such joins, there is a *fact* table and each row from the fact table matches with *at most one row* from the other *dimension* tables (in the presence of predicates on dimension tables). Hence, such joins mirror a predicate over a wider rowset. As discussed for the case of predicates above, a distinct sample of the fact table over the join columns can suffice for such joins.

For many-to-many equi-joins, uniformly sampling the join inputs has both poor performance and accuracy. When multiple inputs are large, sampling only one input has only a small performance improvement. Observe that for a desired sampling fraction over the join output, pushing down uniform random sampling on the join inputs is not performant, as the sampling fraction on each join input will often need to be much larger than the desired sampling fraction over the join output. Further, since a row in one input can match with multiple rows in another input, obtaining a uniform sample of the join result also requires that rows in each input be sampled based on the frequency with which they match in the other tables [11, 34, 35, 38, 41, 58, 64]. The applicability of techniques that have been proposed to support such correlated sampling does not extend to general queries such as when the input to join is a nested SQL statement.

We now consider what kind of sampling operator might help us effectively sample inputs to the join (i.e., push down the sampling operation) in order to obtain a sample of the output of the join. Our proposal for *universe sampler* addresses this need. Universe sampler is a slight relaxation of the uniform sampler and has higher variance but is appealing under many practical cases. Given parameters $\mathcal{C}, p$, the universe sampler picks uniformly at random a p-fraction of the *values of columnset* $\mathcal{C}$ and outputs all rows that have those values. If multiple join inputs apply the same universe sampler on the join columns, i.e., they pick the same random values of the join columns,[2] then universe-sample-then-join is identical to join-then-universe-sample. Note again that the universe sample of join output is not a uniformly random sample; it has a higher variance which can be precisely quantified. However, not only can the universe sampler be pushed down to the inputs of join, it is also performant because only a p fraction sample of inputs is needed to achieve a p fraction sample of the join output. A similar idea has been used earlier to estimate document similarity [20], to estimate join cardinality [81] and to clean stale views [62]; however, the uni-

[1] For a sub-class of query sub-expressions, which will be described precisely in §2.2, a posteriori error estimates can be offered.

[2] this requires some support from the query optimizer.

verse sampler can be used more broadly towards sampling general queries [60].

2.2 Query Optimization over Samplers

While the above sampler operators can be inserted as desired by the user, large performance gains are possible when these operators execute deep in the plan (*e.g.*, universe sampler on join inputs as opposed to sampling the join output). Therefore, a natural question to ask is given a user-specified query with samplers, can we automatically generate plans that are more performant while preserving the accuracy?

Suppose samplers are specified as table valued functions. Consider the following important class of query sub-expressions where the user computes grouped aggregates over a sampled relation.

```
SELECT C, Agg_1, ..., Agg_n
FROM R SAMPLE <samplertype> <samplerparams>
GROUP BY C
```

Here, $\mathcal{C}$ is a set of columns, $\texttt{Agg}_\texttt{i}$ are SUM-like aggregates and most importantly $\mathcal{R}$ can be any general relational expression consisting of, *e.g.*, nested SQL statements. Note that this class of query sub-expressions is much broader than is supported by many AQP techniques [10, 11, 13, 32, 57, 64].

For any query sub-expression in the above class, a query plan transformation rule is said to be *accuracy-preserving* if the following conditions hold: (1) Each group in the query answer has the same likelihood of appearing in both plans. (2) Estimates of aggregates for each group have the same expected value in both plans. (3) The plan produced by the transformation rule has a no worse variance of the estimate of aggregate (compared to the plan before the transformation).

For the above class of query sub-expressions, a recent work shows that the query optimizer can offer more performant plans while offering the same accuracy as the query specified by the user [59]. We briefly sketch the approach below.

If only the uniform sampler is supported, it is easy to show that accuracy is preserved when pushing the sampler below selections, projections and to the fact table in foreign-key equijoins; several prior methods implicitly use this property [10, 11]. If the samplers get pushed on to the base tables, they can also leverage physical access methods (*e.g.*, indices, columnar layouts) [71].

Interaction with the query optimizer is more interesting, however, when multiple samplers are available. The following TPC-DS style query illustrates how interaction between samplers helps pushdown.

```
SELECT COUNT(*) FROM
(SELECT DISTINCT customer_sk
 FROM store_sales, store_returns
 WHERE ss_customer_sk = sr_customer_sk)
SAMPLE UNIFORM 10%
```

The above syntax implies that `COUNT(*)` is to be computed over a sampled relation. Since the join is not a key foreign-key join, pushing the uniform sampler below the join leads to large error (see §2.1 and [34]). However, a 10% universe sampler on `customer_sk` has the same accuracy as the uniform sampler. Because of `SELECT DISTINCT`, we know that each distinct value of `customer_sk` occurs exactly once in the input of the sampler. Hence, a universe sampler which picks 10% of the distinct values of `customer_sk` uniformly-at-random is statistically identical to a uniform sampler which picks 10% of the rows uniformly-at-random. Now, the universe sampler can be pushed below the equi-join. Doing so can reduce cost substantially especially when the join is distributed. The universe sampler can also be pushed down further, in each relation, past selections, projections, foreign-key equijoins and any equijoins on `customer_sk`; all of which can further lower cost.[3]

To facilitate sampler pushdown, as discussed in the above example, we need to identify a set of query transformation rules that *locally* preserve accuracy. However, in many important cases, whether pushing down an accuracy-preserving sampler will improve the plan cost depends on the dataset, as in traditional query optimization. Furthermore, a vast number of alternative plans are available. Hence, a cost-based exploration within the context of QO is appropriate. The following TPC-DS style query illustrates these aspects.

```
SELECT i_color, SUM(ss_sales) FROM
(SELECT * FROM store_sales, item
 WHERE ss_item_sk = i_item_sk)
SAMPLE DISTINCT {{i_color}, 30, 10%}
```

Here, the user may have used the distinct sampler to ensure that at least 30 rows will be sampled for each group (distinct value of `i_color`). Column `ss_item_sk` is a foreign key and `store_sales` is much larger than the `item` table. Hence, pushing the sampler to `store_sales` can improve performance. But, this pushdown appears impossible since the `i_color` column does not appear in `store_sales`. Note however that `i_color` is functionally dependent on the join column `ss_item_sk`; hence pushing the distinct sampler $\{\{\texttt{ss\_item\_sk}\}, 30, 10\%\}$ onto `store_sales` will preserve accuracy. Whether or not such pushdown improves performance, however, depends on the number of distinct values of `ss_item_sk` relative to the number of distinct values of `i_color`. Since the former is larger, gains from pushing the sampler below the join may be offset by the larger sample size. Thus, a cost-based optimization is needed to pick the appropriate plan.

Motivated by the above examples, a recent work considers the rich interaction between samplers in a query optimizer [59]. It offers several sampler pushdown rules that *locally* preserve accuracy. Using these rules the QO explores many alternative plans in the typical cost-based manner.[4] A key point to note is that the new samplers beyond the uniform sampler open up the opportunity for new plan transformation rules; using these rules the QO can push samplers deeper into the query plan while preserving accuracy. Of course, the magnitude of performance gains depend on how deep the samplers can go in the query plan.

2.3 Online Aggregation

Online aggregation is an important alternative approach to the query-time sampling method that we discussed above. Online aggregation methods (*e.g.*, [28, 37, 50, 53, 54, 57, 74]) progressively execute the query on *prefixes* of the input; they present the result to the user and update the result as more input is processed. Under the assumption that every prefix is a uniform random sample of the whole input, confidence intervals for aggregates can be offered for some queries. As more input rows are processed, the confidence intervals shrink. The key advantage of online aggregation is that the user receives an approximate answer quickly and may terminate the query when the confidence levels are satisfactory.

Although online aggregation has an attractive value proposition, unlike the techniques based on samplers described in Section 2.1

[3]The query syntax used here does not show one important aspect, the actual aggregates in the answer are in practice replaced by expressions that are unbiased estimators of these aggregates.

[4]The system also uses pushdown rules that preserve accuracy only in a weaker sense (*e.g.*, when input size is large) because they can lead to large performance gains; refer to the papers for details.

and Section 2.2, online aggregation needs significantly more work to integrate into existing data platforms. First, they need physical access methods that satisfy the criteria that each *prefix* of the input is a uniform random sample. Without such support, random access over tables has high I/O cost. Maintaining the invariant that every *prefix* of the input is a random sample of the input as data evolves is also challenging. Some works address the physical design issues [55, 56, 76]. Next, online aggregation needs special operator implementations to support progressive execution (*e.g.*, ripple join) [50, 57, 64]. Ripple join [50] is a family of operators that progressively compute a join but it requires the inputs to the join to be memory resident for efficiency. This limitation is relaxed by SMS join [54] which divides the overall join into a union over many ripple joins each of which computes a join over portions of the input that are guaranteed to fit within memory. The DBO system [57] improves join operator efficiency further and shows how to leverage indices on input. WanderJoin [64] takes the use of indices to an extreme; it shows that given appropriate indices, a random walk-like method which joins individual rows from the inputs can yield a random sample of the join output. However, these indices could be expensive to create.

3. ACCURACY GUARANTEES USING PRE-COMPUTED SAMPLES

To achieve a much sharper reduction on response time compared to the query-time sampling techniques introduced in Section 2, we can draw samples from the data in a pre-processing step and use them to process incoming queries. This intuition is hardly novel. Indeed, there is a long line of work developed based on this idea, *e.g.*, [11, 10, 31, 17, 32, 33, 72, 79, 13]. Random samples from data rows can be drawn and materialized as sample tables in the database. These pre-computed "small" sample tables are used to answer an incoming query, thus achieving significant reduction in running time of the query.

As mentioned in the introduction, there are no silver bullets. Indeed, AQP systems based on pre-computed samples focus on a subclass of SQL queries. A query in this class is a SQL query with an aggregation on one or more columns, along with a predicate, a group-by operator, and possibly foreign-key joins. Such *aggregation queries* are, however, very common and important in OLAP and many business intelligence applications.

Although AQP systems based on pre-computed samples offer low latency, a key drawback is that such systems typically do not offer a priori accuracy guarantees. This deficiency makes querying such systems "hit or miss" despite significant investments in pre-computation that such systems demand.

In Section 3.1, we review a few representative examples of past work. In Section 3.2, we reflect on the accuracy models provided by such systems and explain why they fall short. In Section 3.3, we discuss what we consider to be a promising approach - the ability to give an accuracy guarantee in a query-independent manner using pre-computed samples.

3.1 Choosing Samples in Pre-processing

There is a trade-off between pre-processing and query-time costs and the accuracy of answers. How the samples are chosen and how large the sample table are both crucial factors in the accuracy and the effectiveness of the technique in reducing of the running time of the query. Deciding the criteria for selecting tuples for inclusion in the sample table is challenging. Consider an ad-hoc query with a predicate. It could be that after the sample table is filtered by the predicate, we may not have sufficiently many sample rows left to accurately estimate the answer to the query. Ideally, our goal is to ensure that sufficiently many sample rows are left in each group, after the predicate in the incoming query is evaluated on the sample table and the group-by is executed, so that the estimated answers have sufficiently small errors. Similarly, the accuracy of estimation for an aggregate such as $\texttt{SUM}(A)$ is sensitive to whether the outlier values of attribute A are in the sample.

Inspired by the stratified sampling technique [65] from statistics, AQP systems calibrate sampling fractions in different parts of a table to create samples to address this challenge. In some AQP systems, such calibration is done purely based on the schema and statistics of the table (*workload-independent*), *e.g.*, [11, 10, 31, 17]. This line of work was initiated by the AQUA system [11, 10], which considers all possible combinations of grouping columns and chooses a different sampling fraction per combination. There is another line of work that utilizes historical workload distributions (*workload-aware*), *e.g.*, [32, 33, 72, 79, 13], with the assumption that future workloads are identical to, or at least, have a similar distribution to the past ones. For example, workloads are used in [46] to construct biased samples. STRAT [32, 33] aims to minimize the expected relative error of the query workload. We now discuss in detail one example workload-independent and workload-aware technique for creating the sample table.

Babcock *et al.* [17] proposes *small group sampling*, a stratified sampling technique that builds both a global uniform sample and a biased sample on each single column. Uniform sampling does a satisfactory job at providing good estimates for the large groups in an aggregation query. To improve the accuracy of aggregations on small groups, for each column, rows with infrequent values in that column are included in the biased sample. A query is executed on the union of a uniform sample and all biased samples whose columns appear in the query to estimate the answer. A shortcoming of this approach is that, in real workloads, small groups may be caused by the intersection of constraints on two or more columns, which cannot be captured by the above set of biased samples.

BlinkDB [14, 13] is a representative workload-aware AQP system, which can handle small groups on multiple columns if these column combinations (or parts of them) are common in the workload. It first abstracts workload information to *the set of columns that appear in a query* (called a *QCS*), so that it can get a meaningful estimation of the workload distribution. For each possible QCS, it can create a stratified sample – similar stratified samples on single columns can be found in Babcock *et al.* [17]. But since it is too expensive to create and maintain stratified samples for all QCSs, BlinkDB formulates an optimization problem to decide how to allocate space budget across samples for different QCSs based on previous workloads to minimize the loss of accuracy from samples for the overall workload distribution.

The inherent deficiency of workload-aware AQP systems is that the workload may keep changing [67], especially for exploratory workloads where AQP would be the most beneficial. Detecting workload distribution changes and re-optimizing samples accordingly can be an expensive recurring task in these systems.

Another goal while building the sample table is to ensure that rows with different values on an aggregation column are captured, as mentioned at the beginning of this subsection. The outlier index proposed in [31] stores rows with outlier values on each column. It is used together with a uniformly random sample to produce unbiased estimations with reduced variances for `SUM` and `AVG`.

3.2 Lack of A Priori Accuracy Guarantees

As queries are answered against pre-computed samples, it is important for an AQP system to provide estimated errors in the an-

swers. There are several ways to measure errors in previous systems. For example, *confidence intervals (CI)* are used in [11, 13]. A CI $= [\text{est} - w, \text{est} + w]$ with 90% confidence level means that, informally, with 90% (pre-sampling) probability, the CI covers the true answer, where est is the estimated answer for a group [65]. *Mean squared error* is another error model used in [32, 17]. Bootstrap [85] is used to provide tighter error estimations in the answers.

No matter which of the above error models is used, a common issue in the accuracy contracts of previous systems is that they can estimate the error in an answer only after the query is processed, but cannot guarantee that the error is below a pre-specified threshold in advance. Let's take the error models based on CIs as an example. They can be easily supported for a *broad class of sampled inputs* and for *complex queries*. CIs are calculated using, *e.g.*, the central limit theorem (CLT) or the Hoeffding inequality [47], and can also be supported for different aggregate functions. However, CIs are *data-dependent*. Namely, when the values on a column A have a large standard deviation or a large range $\texttt{MAX}(A) - \texttt{MIN}(A)$, the resulting CI for a query with an aggregation $\texttt{SUM}(A)$ could be arbitrarily large. This is a critical issue for AQP systems using pre-computed samples: with the sizes of sample tables fixed after pre-processing, the errors or CIs can be either small or large for an incoming query. Thus, the effectiveness of the system is "hit or miss".

3.3 AQP with Query-Independent Accuracy Guarantee

To the best of our knowledge, all previous AQP systems based on pre-computed samples (including those discussed in Section 3.1), succeed in estimating errors for an ad-hoc query a posteriori but fail in bounding errors ($\leq \epsilon$) of every incoming query in the supported query class a priori. The ability to bound the error of an AQP system for every query makes such AQP systems far more attractive. Asking queries on such systems then will have a Service Level Agreement and no longer appear to be a "hit or miss" experience.

Such a strong guarantee cannot be realized for all different aggregates and error models. We will focus on queries with SUM as the aggregation function to illustrate how previous approaches fail in providing this guarantee, and the key innovations in a recent work, Sample+Seek [39], towards having a query-independent accuracy guarantee for a subclass of aggregation queries. Consider the following simple query as an example.

```
SELECT B, SUM(A) FROM T
WHERE C = 10
GROUP BY B
```

• *Better sampling strategy is needed.* If the predicate "WHERE $C = 10$" is not selective, *i.e.*, a large number of rows in T satisfy it, previous approaches like Babcock *et al.* [17] and BlinkDB [13] rely on mainly a uniformly random sample to estimate $\texttt{SUM}(A)$ for each group. Since each row is in the sample with equal probability, the estimation has a large variance (error) if the distribution on A is skewed. Both approaches can estimate the error but the error can be arbitrarily large for a fixed sample size.

The core idea in the sampling strategy used in Sample+Seek is to pick a row with probability proportional to its value on column A. The intuition is that rows with values close to the average of A can be picked with lower probability than the outliers without having a huge impact on the resulting estimation of $\texttt{SUM}(A)$. This strategy can be thought of as a randomized version of the outlier indexing in [31] with an optimal accuracy guarantee. It also generalizes the accuracy analysis of uniform sampling for COUNT in [9].

• *Help from indices needed.* If the predicate "WHERE $C = 10$" is selective, *i.e.*, only a few rows in T satisfy it, a global random sample does not suffice as it may contain none of these rows. We do not need to estimate the selectivity in advance. Instead, we can always process the query with pre-computed samples first; if there are not sufficiently many sample rows satisfying the predicate, we turn to indices in [39] for help. Note that these indices can use the traditional implementation of indices in most data platforms. These indices are used in two ways. In one way, a one-dimensional index can be used to retrieve all the rows satisfying "$C = 10$" – since there are only a few such rows, we can scan them for the exact answer. We refer to such indices as *Low-Frequency Index*. Also, an index may be used to retrieve a random sample of the IDs of rows satisfying "$C = 10$", with the property that the ID of a row appears in this sample with probability proportional to its value on A. In order to draw this sample for a predicate on multiple columns, we need to compute index intersection on multiple one-dim indices; however, we only need a prefix of the index intersection result – the indices can be built in such a way that the prefix gives us the desired sample of row IDs. Only for row IDs in this small sample, we need to perform index seeks to look up values of the rows on columns A and B, in order to estimate $\texttt{SUM}(A)$ for each group.

The biased samples in Babcock *et al.* [17] can also be used in the first way above, but they cannot handle selective predicates on two or more columns, *e.g.*, "$C = 10$ AND $D = 20$", with the same accuracy guarantee. Stratified samples on QCSs in BlinkDB [13] can handle selective predicates on more than one column, but storage constraints may limit the number of multi-dimensional QCSs that can be effectively supported.

• *Accuracy guarantee can be provided.* The samples and their sizes in Sample+Seek are calibrated carefully with the indices, and they together can seamlessly cover both selective and non-selective queries. For COUNT and SUM, Sample+Seek can provide query-independent accuracy guarantee in terms of the *distribution accuracy*: namely, we can normalize both the exact and the estimated answers such that the normalized values of all groups sum up to 1, and the *distribution error* is defined to be the L^2 distance between the two normalized answers. For example, suppose the exact answer to the example query is $\mathbf{x} = \langle 69, 31\rangle$ with two groups on B, and the estimated answer $\hat{\mathbf{x}} = \langle 70, 32\rangle$ is provided; the distribution error in $\hat{\mathbf{x}}$ is $\sqrt{(69/100 - 70/102))^2 + (31/100 - 32/102)^2} \approx 0.005$. It is guaranteed that, for any query in the supported class, the distribution error in the estimated answer is always no more than a pre-specified error threshold ϵ, with high probability.

One can refer to [39] for how to support different types of predicates,[5] *e.g.*, range constraints, on multiple columns, and foreign-key joins across tables. It is also discussed how the accuracy guarantee generalizes for other aggregates, *e.g.*, AVG and STD.

It is an open problem to generalize the strong accuracy guarantee above for other error models, *e.g.*, mean squared error. The formal specification of the AQP problem with accuracy guarantee may be stated as follows: for a user-given error threshold ϵ in a database, pre-compute samples and indices whose total size is a function of the database size, the class of queries supported, the set of columns in aggregates and predicates, and the error threshold ϵ; such that the system ensures that *any incoming query in the supported class can be answered using pre-computed samples and indices with an error at most ϵ, with high probability.*

[5] A technical requirement for the indices in Sample+Seek to work effectively is that the predicates are sargable, which is common in OLAP workloads.

3.4 Comparison with Query-Time Sampling

AQP techniques based on pre-computed samples achieve a sharp reduction in response time in comparison to query-time sampling techniques, but are limited to a subclass of aggregation queries (single-block and with only key foreign-key joins). If the notion of distribution accuracy is satisfactory to the user, Sample+Seek, described above, provides a rigorous and *a priori accuracy guarantee* in a *query-independent* manner. However, such a guarantee requires support for pre-computed sample tables as well as indices. The cumulative costs of maintaining various samples and indices (especially for different aggregation functions and on multiple aggregation columns) may lead to large pre-computation (and maintenance) overhead. Considering this overhead, the ability to compress these samples and indices to reduce the storage overhead while maintaining run-time efficiency is a technical challenge worth addressing. Note also that larger error tolerance helps reduce such overhead. Sample+Seek as well as other approaches that optimize samples in a workload-independent way (*e.g.*, Babcock *et al.* [17]) are better if the workload distribution is unpredictable. Systems which optimize samples based on historical workloads (*e.g.*, BlinkDB [14, 13]) might be a good alternative if the workload distribution is static and the number of columns is not large.

In contrast, query-time sampling, using approaches described in §2.1 and §2.2, does not need pre-computation or maintenance overhead of sample tables (and indices, for Sample+Seek). And, if queries can have nested SQL statements or joins that are not key foreign-key equijoins then current pre-computation based techniques cannot be used and query-time sampling remains the only alternative. However, query-time sampling may have smaller performance gains unless samplers can be pushed deep into the query plan and unless there exist physical access methods on base tables when sampler operators are pushed down to those base tables. Furthermore, unlike Sample+Seek, no accuracy guarantee is provided at the query level. Query-time sampling also blends with existing Data Platforms most seamlessly and additional sampling operators can be added by using the extensibility mechanisms. However, online aggregation, a variant of query-time sampling, has higher engineering cost of integration with existing data platforms.

4. RELATED WORK

There is such a vast body of work in approximate query processing. In fact, a broader definition of approximate query processing could also include techniques such as entity matching and de-duplication. However, in this paper (and in this section on related work), we consider only approximate query processing from the perspective of answering queries with significant reduction in work or latency with only low error in estimated answers.

SnappyData [78, 5] supports approximate answers in a streaming, transactional and interactive system. It reuses many findings from BlinkDB [13] and some lessons are summarized in [68]. SparkSQL [6] supports online aggregation and so-called delta update queries with bootstrap-based error bounds [84]. Beyond these instances, approximate query processing has not gained wide adoption in data platforms.

While several research works consider issues in sampling over streams [19, 36, 18], sampler operators are not well-supported in streaming engines (*e.g.*, Trill [27], Spark Streaming [83]) with one notable exception [5], as discussed above. Sampling may help because these engines are memory constrained and, unlike sketches, samples can support more general queries.

In terms of a priori guarantees on error, only `COUNT` can be supported by uniform or stratified samples. `SUM`, `AVG`, and `STD` are supported using the measure-biased sampler [39] and GEE [29] can support `DISTINCT COUNT`. A posteriori estimates of error are easier to offer; for `COUNT`, `SUM`, `AVG`, and `STD` and for any generalized uniform sampler, it is possible to compute a posteriori variance-based bounds and tail probability bounds [70]; similar support is also possible for the distinct and universe sampler. Handling aggregates on arbitrary arithmetic expressions and user-defined functions is more challenging in general and many problems remain open.

To support an accuracy latency tradeoff, BlinkDB [13] uses error-latency profiles (ELP) to choose among multiple pre-computed samples. The ELP of a query records the observed accuracy and latency when a query is executed on a sample; it is constructed by executing the query on all pre-computed samples. The key challenge they address is: when a new query arrives with an accuracy and performance target, how to consult the ELPs to find an appropriate sample (if any)? In the context of query-time sampling, one can apply sampler pushdown rules that *locally* trade-off a small degradation in accuracy for better performance [59].

To improve costing inside query optimizers, samples are used to quickly estimate the cardinality of query sub-expressions [81, 63, 41]. Cardinality estimation is akin to a `COUNT`$(*)$ query without group. Hence, even though these techniques are related to AQP they do not generalize to query answering.

Nirkhiwale *et al.* [70] show how to compute unbiased estimators of `SUM`-like aggregates and variance of estimators for plans with arbitrarily many generalized uniform sampler (GUS) operators. The key idea is to transform plans with arbitrarily many samplers to a new plan that has a single GUS operator just before the aggregate such that the two plans are *equivalent*. This transformation helps derive closed form expressions for the unbiased estimator and the estimator's variance. The above insight is extended in [60] to consider operators that are not GUS (*e.g.*, the distinct and universe sampler in §2.1) and to analyze the likelihood of missing groups. It remains an open issue to extend this analyses method for other samplers (*e.g.*, non GUS, not distinct and not universe). Alternatively, one can use the bootstrap operator over samples to estimate plan accuracy [12, 85, 84].

In the rest of this section, we discuss other approaches to approximate query processing. View materialization [52, 15, 49] techniques pre-compute answers to some queries and create summaries to accelerate OLAP and decision-support queries exactly (no approximation). Several engines use a subset of these techniques [1, 8, 7]. Numerous efforts reduce the sizes of data cubes on high-dimensional data; see [51] for a survey. Similarly, histograms [48], wavelets [82, 26, 48] and sketches (discussed below) can be used to compute some aggregates approximately; see [47] and [38] for good summaries of these techniques. The intuition is that these data structures act as synopses of the original data and suffice to approximately answer a subclass of queries. Their key advantage is that for some queries, these techniques give better answers than sampling-based systems (*e.g.*, exact, or estimates with a small variance). The common drawbacks are that they cannot support general queries; they have a large space overhead for high-dimensional datasets and the space overhead becomes even larger to support selections and group-by's.

Sketches are a specific type of data synopses [38, 40, 45] that has had a huge impact on data stream processing [69]. Sketches are specific to particular aggregates and can be computed in one pass over the data. Once computed, the sketch can answer queries on the corresponding aggregate without having to examine the raw inputs. Because sketches gain compactness and computational efficiency at the cost of information loss, after a sketch is constructed for one

aggregate or predicate, the sketch cannot be used to support queries having other aggregates or predicates.

Another line of work [44, 43, 24, 42, 22] answers a specific class of queries by accessing only a bounded number of data rows with the help of indices built on application-induced cardinality constraints. The BEAS system [23] uses a similar idea for AQP. A recent work [77] proposes a deterministic approximate querying scheme. It uses novel bit-sliced indices and evaluates a query on the first few bits of all the rows as an approximation. Effectiveness of such an approach for applications need to be studied.

5. WHERE TO GO FROM HERE?

As mentioned in the introduction, despite much technical progress, approximate query processing has not gone mainstream. As a community, we can continue to make more technical progress but in our opinion, we should pause and ask ourselves what it will take to make approximate query processing real. In this paper, we have argued that there is no silver bullet. In other words, there does not appear to be a single technical challenge, solving which would then realize the potential of approximate query processing. We have argued that we must ask ourselves what clear value proposition can we offer to users of an approximate query processing system? Based on that analysis, we have proposed a two-pronged approach. On one hand, we should equip an application programmer with important primitives in the query language so that they can write application logic that does application-specific approximation. This motivated our work on query-time sampling (§2). On the other hand, for a restricted query language but one that caters to business intelligence and OLAP queries, we can provide accuracy guarantee in a query-independent way, using pre-computed samples and indices (§3). By sharply reducing the latency for ad-hoc queries along with an accuracy guarantee, data analysts can have a new way to interact with the system for data exploration queries. Beyond these two approaches, we as a community need to envision alternatives where approximate query processing can offer high value to users.

Beyond pursuing clear value propositions for approximate query processing as discussed above, we offer these recommendations:

Integrate AQP with data platforms: AQP techniques that can be retrofitted easily into existing data platforms will be able to ride the curve on the engine improvements such as vectorization, columnar layouts, and architectural trends. With its integration with data platforms, AQP techniques will have a greater chance of reaching a broad base of users. In contrast, stand-alone systems will have to meet a very high bar both in terms of performance and the required support for querying functionality before they are widely used. Data lake or data warehouse systems present a particularly promising opportunity for AQP. Because queries on such platforms can consume enormous amounts of cluster hours (*e.g.*, log analysis pipelines over click-logs or crawl-logs or cluster-logs), even if query-time sampling offers a small relative reduction in cost (*e.g.*, 50%), these savings may be large in absolute terms. Furthermore, pre-computed samples could be used to support an *interactive mode* on large datasets in data lake systems; even though the query space is restricted, users may be enticed by the low latency responses and accuracy guarantees only.

Approximate execution mode for querying: Exposing an approximate mode for queries, especially for interactive queries, will provide a powerful way to safely experiment with the effectiveness of AQP systems and thus over time overcome the resistance to accept approximate answers. We envision an experimentation mode where whenever a data analyst submits an ad-hoc query, the system will compute both the true answer and the answer using the AQP system *concurrently*. If the AQP technique can respond quickly, the result may be presented as a quick-but-imprecise preview of the full answer inviting the user to provide one of two responses: "continue to answer the full query" or "cancel the full query". Such a setting will allow us to better evaluate the effectiveness of the AQP systems by obtaining implicit user feedback. Ideally, such a mode can be targeted at queries that are more likely to benefit. Unlike online aggregation which offers a continuously changing answer, the approximate mode will offer only two possible answers; one from executing the query fully and another from the AQP plan. Furthermore, unlike online aggregation, the approximate mode does not need new operator implementations (as mentioned in §2.3). Such an approach to evaluation can also be used for batch queries where AQP can reduce resource usage.

Experiment with new scenarios: Interactive exploration over data is perhaps the most direct application for AQP, as discussed. Here, the queries could be relatively simple (*e.g.*, only select-project-join-groupby operations) and the user may not need a precise answer but cares deeply about extremely quick responses [21]. Another use case is that of production query pipelines; we see many periodically recurring queries in our big data clusters that analyze logs to compute KPIs for human consumption or for visualization dashboards. The inputs to such queries are enormous *e.g.*, click logs, server access logs. The outputs, restricted by human cognition and screen sizes, are rather small. Hence, such queries typically have groups and aggregates and AQP may reduce cost. However, these queries also tend to be non-trivial (*e.g.*, nested SQL statements). Yet another important scenario is the challenge of extracting a sample efficiently from a back-end batch processing system so that the sample can be reused repeatedly for experimentation by data analysts on their desktop. New scenarios also bring with them possibly different kinds of accuracy requirements. For example, the user may only care about the rank order of the result [61] or the user may only want to preserve the labels that are output by a machine learning classifier. How best to support different kinds of accuracy requirements for AQP remains an open question.

Approximate query processing started with a very attractive value proposition. It is even more attractive today with the data deluge. But unless we rethink our approach to approximate query processing with an eye towards scenarios and clear value propositions, we will have only technical results, not usable systems.

Acknowledgements

We are deeply thankful to Christian Konig and Vivek Narasayya for reading numerous versions of this document; their feedback has improved this report substantially.

6. REFERENCES

[1] Microsoft powerbi. https://powerbi.microsoft.com/en-us/.

[2] Oracle data mining blog: To sample or not to sample. https://blogs.oracle.com/datamining/entry/to_sample_or_not_to_sample.

[3] Sampler in oracle sql server. http://bit.ly/2n7TZow.

[4] Sampling in google bigquery. https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#approx_top_sum.

[5] SnappyData.IO. http://www.snappydata.io.

[6] Sparksql support for continuous answers with error bars. https://www.slideshare.net/SparkSummit/agarwal-zeng.

[7] Sql server analysis services. https://technet.microsoft.com/en-us/library/ms175609(v=sql.90).aspx.

[8] Tableau. https://www.tableau.com/products/cloud-bi.

[9] J. Acharya, I. Diakonikolas, C. Hegde, J. Z. Li, and L. Schmidt. Fast and near-optimal algorithms for approximating distributions by histograms. In *PODS*, 2015.

[10] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD*, 2000.

[11] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, 1999.

[12] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[13] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eurosys*, 2013.

[14] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 5(12), 2012.

[15] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[16] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark sql: Relational data processing in spark. In *SIGMOD*, 2015.

[17] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

[18] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, 2002.

[19] A. Bagchi, A. Chaudhary, D. Eppstein, and M. T. Goodrich. Deterministic sampling and range counting in geometric data streams. *ACM Trans. Algorithms*, 2007.

[20] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*. IEEE Computer Society, 1997.

[21] J. Brutlag. Speed matters for Google web search. http://bit.ly/1b4RKoZ, 2009.

[22] Y. Cao and W. Fan. An effective syntax for bounded relational queries. In *SIGMOD*, 2016.

[23] Y. Cao, W. Fan, and C. Hu. Data driven approximation with bounded resources. *PVLDB*, 10, 2017.

[24] Y. Cao, W. Fan, T. Wo, and W. Yu. Bounded conjunctive queries. *PVLDB*, 7(12), 2014.

[25] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive Datasets. In *VLDB*, 2008.

[26] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDBJ*, 10(2-3), 2001.

[27] B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. Platt, J. Terwilliger, J. Wernsing, and R. DeLine. Trill: A high-performance incremental query processor for diverse analytics. In *VLDB*, 2015.

[28] B. Chandramouli, J. Goldstein, and A. Quamar. Scalable progressive analytics on big data in the cloud. In *VLDB*, 2014.

[29] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, 2000.

[30] S. Chaudhuri. What next?: a half-dozen data management research goals for big data and the cloud. In *PODS*, 2012.

[31] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, 2001.

[32] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD*, 2001.

[33] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 32(2), 2007.

[34] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *SIGMOD*, 1999.

[35] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD*, 1998.

[36] K.-T. Chuang, H.-L. Chen, and M.-S. Chen. Feature-preserved sampling over streaming data. *ACM Trans. Knowl. Discov. Data*, 2009.

[37] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, 2010.

[38] G. Cormode, M. Garofalakis, P. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 2012.

[39] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, 2016.

[40] M. Durand and P. Flajolet. Loglog counting of large cardinalities (extended abstract). In *ESA*, 2003.

[41] C. Estan and J. F. Naughton. End-biased samples for join cardinality estimation. In *ICDE*, 2006.

[42] W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In *PODS*, 2015.

[43] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.

[44] W. Fan, X. Wang, and Y. Wu. Querying big graphs within bounded resources. In *SIGMOD*, 2014.

[45] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *DMTCS*, 2007.

[46] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, 2000.

[47] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[48] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *PODS*, 2001.

[49] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, 1(1), 1997.

[50] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD*, 1999.

[51] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[52] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[53] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *SIGMOD*, 1997.

[54] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. The sort-merge-shrink join. *ACM Trans. Database Syst.*, 2006.

[55] C. Jermaine, A. Pol, and S. Arumugam. Online maintenance of very large random samples. In *SIGMOD*, 2004.

[56] C. M. Jermaine. Online random shuffling of large database tables. *IEEE Trans. Knowl. Data Eng.*, 19(1):73–84, 2007.

[57] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the DBO engine. In *SIGMOD*, 2007.

[58] N. Kamat and A. Nandi. Perfect and maximum randomness in stratified sampling over joins. *CoRR*, abs/1601.05118, 2016.

[59] S. Kandula. Errata and proofs for "quickr". Technical Report TR-2017-14, MSR, 2017.

[60] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.

[61] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5), 2015.

[62] S. Krishnan, J. Wang, M. Franklin, K. Goldberg, and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. In *VLDB*, 2015.

[63] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD*, 2007.

[64] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *SIGMOD*, 2016.

[65] S. L. Lohr. *Sampling: Design and Analysis*. Thomson, 2009.

[66] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. In *VLDB*, 2010.

[67] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. Cliffguard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[68] B. Mozafari and N. Niu. A handbook for building an approximate query engine. In *IEEE Data Engineering Bulletin*, 2015.

[69] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[70] S. Nirkhiwale, A. Dobra, and C. Jermaine. A sampling algebra for aggregate estimation. In *PVLDB*, 2013.

[71] F. Olken. *Random Sampling from Databases*. PhD thesis, UCBerkeley, 1993.

[72] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive analysis of web-scale data. In *CIDR*, 2009.

[73] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[74] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11), 2011.

[75] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD*, 1984.

[76] A. Pol, C. M. Jermaine, and S. Arumugam. Maintaining very large random samples using the geometric file. *VLDB J.*, 17(5):997–1018, 2008.

[77] N. Potti and J. M. Patel. Daq: a new paradigm for approximate query processing. *PVLDB*, 8(9), 2015.

[78] J. Ramnarayan, B. Mozafari, S. Wale, S. Menon, N. Kumar, H. Bhanawat, S. C. Y. Mahajan, R. Mishra, and K. Bachhav. Snappydata: A hybrid transactional analytical store built on spark. In *SIGMOD*, 2016.

[79] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, 2011.

[80] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive- a warehousing solution over a map-reduce framework. In *VLDB*, 2009.

[81] D. Vengerov, A. Menck, M. Zait, and S. Chakkappen. Join size estimation subject to filter conditions. In *VLDB*, 2015.

[82] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.

[83] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *SOSP*, 2013.

[84] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.

[85] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.