Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee

Bolin Ding[1] Silu Huang[2]* Surajit Chaudhuri[1] Kaushik Chakrabarti[1] Chi Wang[1]
[1]Microsoft Research, Redmond, WA
[2]University of Illinois, Urbana-Champaign, IL
{bolind, surajitc, kaushik, chiw}@microsoft.com, shuang86@illinois.edu

ABSTRACT

Data volumes are growing exponentially for our decision-support systems making it challenging to ensure interactive response time for ad-hoc queries without increasing cost of hardware. Aggregation queries with Group By that produce an aggregate value for every combination of values in the grouping columns are the most important class of ad-hoc queries. As small errors are usually tolerable for such queries, approximate query processing (AQP) has the potential to answer them over very large datasets much faster. In many cases analysts require the distribution of (group, aggvalue) pairs in the estimated answer to be guaranteed within a certain error threshold of the exact distribution. Existing AQP techniques are inadequate for two main reasons. First, users cannot express such guarantees. Second, sampling techniques used in traditional AQP can produce arbitrarily large errors even for SUM queries. To address those limitations, we first introduce a new precision metric, called *distribution precision*, to express such error guarantees. We then study how to provide fast approximate answers to aggregation queries with distribution precision guaranteed within a user-specified error bound. The main challenges are to provide rigorous error guarantees and to handle arbitrary highly selective predicates without maintaining large-sized samples. We propose a novel sampling scheme called *measure-biased sampling* to address the former challenge. For the latter, we propose two new indexes to augment in-memory samples. Like other sampling-based AQP techniques, our solution supports any aggregate that can be estimated from random samples. In addition to deriving theoretical guarantees, we conduct experimental study to compare our system with state-of-the-art AQP techniques and a commercial column-store database system on both synthetic and real enterprise datasets. Our system provides a median speed-up of more than 100x with around 5% distribution error compared with the commercial database.

1. INTRODUCTION

Enterprises are collecting large volumes of data and are increasingly relying on analysis of the data to drive their businesses. For example, data analysts in a retail enterprise slice and dice their sales data to understand the sales performance along different dimensions like product and geographic location with a varying set of filtering conditions. Interactive query response time is critical in such data exploration; studies in human-computer interaction show that the analyst typically loses the analysis context if the response time is above one second [26]. Unfortunately, as data volume (*i.e.*, number of rows) and complexity of filtering conditions increase (*e.g.*, arbitrary user-given logical expression on multiple dimensions), the response time to answer such queries becomes longer unless additional hardware resources are leveraged. Is it possible to fend off the need for additional resources by leveraging the fact that "small" errors are acceptable for such exploratory queries?

For arbitrary SQL queries, it is hard to take advantage of the flexibility of answering queries with "small" errors, because of the difficulty of defining "error" in an easy-to-understand manner. Fortunately, there is a very important class of queries for which we can both crisply formalize the notion of "small" errors and take advantage of this flexibility in bringing down response time using *approximate query processing* (AQP) techniques. A query in this class is a SQL query with aggregation on a few *measure* attributes (*e.g.*, sales and population) along with a filter predicate and a group-by operator (see Section 2.1). Such *aggregation queries* are very common in OLAP and many business intelligence (BI) applications.

Limitations of existing AQP systems. For many tasks, analysts require the returned distribution of (group, aggvalue) pairs to be within a certain error threshold of the exact distribution. Suppose the analyst wants to visualize the (group, aggvalue) pairs using a pie-chart. She would require the error between the two distributions to be small such that the pie-chart based on the returned answer is similar to that based on the exact answer in terms of relative areas for the different groups. Existing sampling-based AQP systems, *e.g.*, in [3, 12, 7, 5], fall short in such tasks for two main reasons:

• *Inadequate semantics of precision metrics*: *Confidence interval (CI)* [3, 5] used by existing AQP systems are inadequate to express the above guarantees. A 90% confidence interval CI $= [\mathrm{est} - w, \mathrm{est} + w]$ means that, informally, with 90% probability (pre-sampling), the resulting CI covers the true value, where est is the estimated value for a group [22]. CI-based techniques measure the error for each group independently, but *not* for the entire distribution: a 90% CI for every group does not mean they are correct for all groups at the same time with 90% probability. What we seek is a precision metric that captures errors across groups. While *mean squared (relative) error (MSE)* [12, 7] provides another alternative to measure precision, MSE can be estimated after the query has been evaluated but cannot be guaranteed in advance.

• *Unbounded (data-dependent) errors*: Consider a SUM query. For a sample S_m of m rows drawn from n rows (in one group) on the

*Work done while visiting Microsoft Research

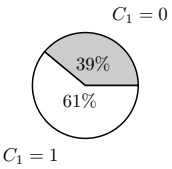
SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

ISBN 978-1-4503-3531-7/16/06…$15.00

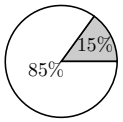
DOI: http://dx.doi.org/10.1145/2882903.2915249

ID	C_1	C_2	C_3	M
1	0	0	0	1
...	0	0	0	1
90	0	0	0	1
91	0	1	0	10
...	0	1	0	10
100	0	1	0	10
101	1	0	0	1
...	1	0	0	1
198	1	0	0	1
199	1	0	0	100
200	1	0	1	100

(a) Example table T: rows 1-90 / 91-100 / 101-198 have the same dimension values



(b) Distribution answer



(c) Case I

(d) Case II

Figure 1: Table and Answer to aggregation query

measure attribute, the CI width is proportional to $\mathrm{std}(S_m)/\sqrt{m}$ [16, 5] for uniform sampling and stratified sampling – $\mathrm{std}(S_m)$ is the *sample standard deviation* on the measure attribute. In particular, for SUM, the 90% CI width is $w \approx 1.65n \cdot \mathrm{std}(S_m)/\sqrt{m}$ (the constant 1.65 is determined by the confidence level 90% and central limit theorem). This result has a profound implication. Since $\mathrm{std}(S_m)$ depends on data and can be arbitrarily large, the CI width can be arbitrarily large. As a consequence, it is not possible to support AQP with a user-specified error bound in many situations.

We illustrate the above drawbacks using an example.

EXAMPLE 1.1. *Consider a table T with C_1 (state), C_2 (product), C_3 (customer group), a measure M (sales), and 200 rows, in Figure 1(a). An analyst wants to find sales number per state with*

Q: SELECT C_1, SUM(M) FROM T GROUP BY C_1.

The answer is a set of (group, aggvalue)-pairs: {(C_1 = 0, 190), (C_1 = 1, 298)}. We can normalize the answer as a distribution $\mathbf{x} = \langle 190/488, 298/488\rangle = \langle 0.39, 0.61\rangle$ *(pie-chart in Figure 1(b)).*

Suppose a sample of 20 rows are drawn uniformly from T to answer Q. With high chance, 9 rows from rows 1-90 with $M = 1$ and 1 row from rows 91-100 with $M = 10$ are picked for the first group ($C_1 = 0$). For the second group ($C_1 = 1$), there are two cases with high probability. Case I: when exactly one of rows 199 and 200 is in the sample, the answer is estimated as {($C_1 = 0$, $19/0.1$), ($C_1 = 1$, $109/0.1$)}, which can be normalized as $\langle 19/128, 109/128\rangle = \langle 0.15, 0.85\rangle$ (pie-chart in Figure 1(c)). Case II: when neither of rows 199 and 200 is taken, the answer is estimated as {(0, $19/0.1$), (1, $10/0.1$)} and normalized as $\langle 19/29, 10/29\rangle = \langle 0.66, 0.34\rangle$ (pie-chart in Figure 1(d)). This shows that uniform sampling can produce large errors. Stratified sampling in [5, 3] and small group sampling in [7] cannot help here. They use different sampling rates for groups of different sizes. But the two groups in the answer to Q have the same size. These techniques end up with drawing uniform samples from each group, and suffer from the same issue.

As mentioned above, for SUM, we have the CI width $w = 1.65n \cdot \mathrm{std}(S_m)/\sqrt{m}$. For the second group in the above ($C_1 = 1$), the true value is 298. In Case I, we have $\mathrm{est} = 1090$, and $w \approx 1470$ – CI correctly covers the true value but is too wide as $\mathrm{std}(S_m)$ is large. In Case II, we have $\mathrm{est} = 100$, and $w = 0$ (as $\mathrm{std}(S_m) = 0$), which does not cover the true value. In both cases, it is difficult to guarantee the error to be within a user-given threshold.

Contributions. In this paper, we build an AQP system to answer aggregation queries with interactive speed that addresses the above two limitations. It *processes aggregation queries with precision of answers guaranteed within a user-given error bound*. We first focus on queries on single table with predicates and COUNT/SUM aggregates, which are central to many BI tasks. We also describe how our techniques are extended for queries with foreign-key joins on multiple tables, and a large class of predicates and aggregates.

Our approach, referred to as *sample+seek* is driven by several key insights. First, we want to find a way to capture precision of the entire distribution across groups. To address this issue, we introduce the notion of *distribution precision*. Second, we assert that AQP is not effective with unbounded data-dependent errors, as illustrated in Example 1.1. Therefore, we must look beyond traditional uniform/stratified sampling-based techniques to find an effective sampling schema aided by indexes if necessary, which is not subject to unbounded errors and able to deal with a large class of queries, including those with highly selective predicates. To address the second issue, we introduce a combination of novel sampling strategy (*measure-biased sampling*) and indexes (*measure-augmented inverted index* and *low-frequency group index*). We describe these three key aspects of our framework below.

• *Precision metric.* We propose to use, *distribution precision*, to measure the precision of the entire distribution over all groups (as opposed to the precision of individual groups) and to allow the analyst to express the desired guarantees. Distribution precision is defined to be the L^2 distance between normalized distributions of the approximate answer and the exact one. In Example 1.1, the distribution answer $\mathbf{x} = \langle 0.39, 0.61\rangle$. And if we give an approximation $\hat{\mathbf{x}} = \langle 0.40, 0.60\rangle$, their L^2 distance is $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 = \sqrt{0.01^2 + 0.01^2} = 0.014$. The analyst specifies the value of a *single parameter*, error upper bound ϵ, and our system *guarantees* to produce an approximation $\hat{\mathbf{x}}$ s.t. with high probability (*e.g.*, 0.9), $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$.

• *Measure-biased sampling.* We address the challenge of avoiding data-dependent error bound by developing a novel sampling scheme called *measure-biased sampling*. The insight is to pick a row with probability proportional to its value on the measure attribute. For SUM-queries without predicates, we show that a *measure-biased sample* of $\mathrm{O}\big(1/\epsilon^2\big)$ rows suffices to give answers under our distribution precision guarantee with errors independent of the (sample) standard deviation of measure values.

• *Indexes for selective predicates.* Large samples have to be maintained to handle queries with highly selective predicates. We address this challenge by building two auxiliary indexes, *measure-augmented inverted index* and *low-frequency group index*, to aid in-memory samples. For selective predicates, we identify rows that satisfy the predicate by looking up these two indexes instead of maintaining prohibitively large samples in memory. The insight is that since we need to lookup rows only for highly selective predicates, we need very few data accesses to guarantee the distribution precision. In particular, we need at most $\mathrm{O}\big(1/\epsilon^2\big)$ random I/O's to answer selective queries under our precision guarantee.

Extensibility. Our solution enjoys *the same degree of extensibility* as other sampling-based AQP techniques (*e.g.* [7, 5]) do and thus supports a range of aggregate functions like AVG, STDEV, and COUNT(DISTINCT $\cdot$). Specialized summaries like views [6], data cube [18], and histograms [17] do not have such extensibility. Our solution can be easily extended for foreign key joins in a star/snowflake schema. The predicates we can handle range from (dis)conjunctions of constraints on categorical dimensions to range constraints and arbitrary AND-OR logical expressions of them.

Our experiments show that compared to the state of the art AQP systems, our sample+seek solution provides answers with equal or less error in orders of magnitude faster response time.

Organization. Section 2 introduces the class of aggregation queries our system can handle and formalizes our distribution precision guarantee. Section 3 overviews our sample+seek approach and system architecture. Details about the "sample" part, *i.e.*, how to

construct and utilize uniform and measure-biased samples, is introduced in Section 4, and the "seek" part, *i.e.*, auxiliary indexes, is in Section 5. Section 6 discusses extensions of our approach. Experimental study is reported in Section 7, followed by related work in Section 8 and conclusion in Section 9. Proofs of most non-trivial theorems and additional experiments are in the appendix.

2. PRELIMINARIES

We first define the class of queries that we focus on, and formalize the precision guarantee our system offers for query answers.

2.1 Single-Block Aggregation Queries

$T(D_1, \ldots, D_{d_C}, M_1, \ldots, M_{d_M})$ is a table with $d_C + d_M$ dimensions. $D_1, \ldots, D_{d_C}$ are group-by and predicate dimensions and $M_1, \ldots, M_{d_M}$ are measure attributes. Types of dimensions are defined by data owner. A column can be both a group-by/predicate dimension and a measure at the same time in different queries.

For a row $t \in T$, let t_D and t_M be the values of dimension D and measure M on this row, respectively, and t_{ID} denote the ID of a row. Let $|T| = n$ be the number of rows in T.

Table aggregation queries. For the ease of explanation, we first focus on queries in the form of $Q(\mathcal{G}, F(M), \mathcal{P})$, called *table aggregation queries*, on a single table T to introduce our techniques:

$$\mathsf{SELECT}\ \mathcal{G},\ F(M)\ \mathsf{FROM}\ T\ \mathsf{WHERE}\ \mathcal{P}\ \mathsf{GROUP\ BY}\ \mathcal{G}. \quad (1)$$

- *Group-by dimensions* $\mathcal{G} \subseteq \{D_1, \ldots, D_{d_C}\}$ are a subset of dimensions of T on which rows are aggregated on.
- *Aggregate measure* $F(M)$ is $\mathsf{COUNT}(*)$ or $\mathsf{SUM}(M)$ where $M \in \{M_1, \ldots, M_m\}$ is a measure attribute.
- *Predicate* $\mathcal{P}$ consists of equality constraints on categorical dimensions $\{D_1, \ldots, D_{d_C}\}$, in the form of "$D_i = v_i$". $\mathcal{P}$ is an AND-OR expression of multiple such constraints.

For simplicity of discussion, we assume that values of a measure M are non-negative when presenting our techniques. How to handle measures with negative values will be introduced in Section 6.1 .

Single-block aggregation queries. Our techniques can be extended for a more general class, *single-block aggregation queries*, which generalizes the class of table aggregation queries in three aspects: i) **(schema)** to support foreign key joins on a star/snowflake schema; ii) **(predicates)** to support range constraints on numeric dimensions and arbitrary AND-OR logical expressions $\mathcal{P}$ of equality constraints and range constraints; iii) **(aggregates)** support more aggregates, *e.g.*, $\mathsf{AVG}(M)$, $\mathsf{STDEV}(M)$, and $\mathsf{COUNT}(\mathsf{Distinct}\ M)$. We introduce how our system handles the above extensions in Section 6.2. The class of aggregates supported by us is the same as the class supported by state of the art AQP systems like BlinkDB [5].

Answers and distributions. An output to an aggregation query Q is a set of (group, value)-pairs, $\{(g_1, z_1), (g_2, z_2), \ldots, (g_r, z_r)\}$, where r is the number of distinct values on the product of group-by dimensions $\mathcal{G} = \{G_1, \ldots, G_l\}$. Each $g_i \in G_1 \times G_2 \times \ldots G_l$ is said to be a *group*, and z_i is the corresponding aggregate value. We can present the output to be an r-dim vector $\mathbf{z} = \langle z_1, \ldots, z_r \rangle$.

The query output $\mathbf{z}$ can be *normalized as a distribution* on all groups $G_1 \times \ldots \times G_l$: let $x_i \leftarrow z_i / \sum_{j=1}^{r} z_j$, for $i = 1, 2, \ldots, r$. The vector $\mathbf{x} = \langle x_1, \ldots, x_r \rangle$ is said to be a *distribution answer* (or *answer* for short) to the query Q. Our guarantee about precision introduced next is w.r.t. the distribution answer with the goal of preserving the relative ratio/order of groups in the query output.

2.2 Distribution Precision Guarantee

Computing the exact output or distribution answer $\mathbf{x}$ to a query Q can be expensive on large tables. Our system produces an estimated distribution answer $\hat{\mathbf{x}} = \langle \hat{x}_1, \ldots, \hat{x}_r \rangle$. We propose to measure the *error* in $\hat{\mathbf{x}}$ using the L^2*(Euclidean)-distance* between $\mathbf{x}$ and $\hat{\mathbf{x}}$, *i.e.*, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 = \sqrt{\sum_{i=1}^{r} (x_i - \hat{x}_i)^2}$. For the ease of understanding, we can rephrase $\|\mathbf{x} - \hat{\mathbf{x}}\|_2$ in words as follows:

$$\sqrt{\sum_{\text{group } i} \left(\frac{\text{group } i\text{'s value}}{\text{total group value}} - \frac{\text{estimated group } i\text{'s value}}{\text{total est. group value}} \right)^2}. \quad (2)$$

Our system allows users to specify an error bound ϵ which is *independent* on both the query Q and the data distribution in table T. The estimated answer $\hat{\mathbf{x}}$ is called an *ϵ-approximate answer* or *ϵ-approximation* of the true distribution answer $\mathbf{x}$ if $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$.

System task and precision guarantee. Our AQP system processes single-block aggregation queries and promises *ϵ-distribution precision guarantee*: that is, for a user-specified error bound ϵ, our system can produce an estimated answer $\hat{\mathbf{x}}$ for any given single-block aggregation query Q, s.t. $\hat{\mathbf{x}}$ must be an ϵ-approximation of the exact answer $\mathbf{x}$ to Q, *i.e.*, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$, with high probability.

As discussed in Section 1, compared to CI, our distribution precision guarantee based on L^2-distance is more rigorous, easier to be interpreted, and has only one parameter ϵ that consistently measures how two answers differentiate globally.

Another good property of our guarantee is: $\max_i |x_i - \hat{x}_i| \leq \|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$. It says that, when ϵ-distribution precision guarantee holds, the max of errors $|x_i - \hat{x}_i|$ for all groups is also bounded by ϵ, with high probability. It implies that the order of groups or the top-k groups derived from $\hat{\mathbf{x}}$ can be wrong by at most ϵ.

3. SYSTEM OVERVIEW

We give an overview of our solution in Section 3.1, including sample/index building and aggregation query processing, and summarize the main theoretical results we obtain. In Section 3.2, we introduce the architecture of our system.

3.1 Overview of Our Approach

To tackle the inherent deficiency of sampling-based approach when handling queries with very selective predicates, we propose a novel *sample+seek* framework. We classify queries into *large queries* and *small queries* based on how many rows satisfy the predicates. For large queries, with many rows satisfying their predicates, we are able to collect enough number of rows satisfying their predicates from a *uniform sample* or a new sample designed for SUM aggregates, called *measure-biased sample*, to obtain ϵ-approximate answers. For small queries, random samples are not sufficient, so we propose two new index structures, called *measure-augmented inverted index* and *low-frequency group index*, to evaluate them for ϵ-approximations using index *seeks*.

3.1.1 Classification of Queries

We classify queries using *(measure) selectivity* into *large queries* and *small queries*, based on the hardness of being answered.

DEFINITION 1. (Selectivity and Measure selectivity) *The* selectivity *of predicate $\mathcal{P}$ in a query Q against a table T is $s(\mathcal{P}) = |T_{\mathcal{P}}|/|T|$ where $|T|$ is the number of rows in T and $T_{\mathcal{P}} \subseteq T$ is the set of rows that satisfy the predicates $\mathcal{P}$ in T.*

The measure selectivity *of predicate $\mathcal{P}$ in a query Q against a table T on measure M is $s_M(\mathcal{P}) = M(T_{\mathcal{P}})/M(T)$, where $M(T_{\mathcal{P}}) = \sum_{t \in T_{\mathcal{P}}} t_M$ and $M(T) = \sum_{t \in T} t_M$.*

DEFINITION 2. (Large queries and small queries) *For a selectivity threshold s_0, a query Q with* COUNT *aggregate is a s_0-*large query *iff $s(\mathcal{P}) \geq s_0$ and otherwise a s_0-*small query.

*Similarly, a query Q with $\mathsf{SUM}(M)$ is a s_0-*large query *iff we have $s_M(\mathcal{P}) \geq s_0$, and otherwise a s_0-*small query.

```
Input:  aggregation query Q(G, F(M), P)
Output: ε-approximate answer distribution
1: If F = COUNT then
2:     (x̂, sup_x̂) ← ProcessWithDataBubble(Q, T^0);
3: Else If F = SUM then
4:     (x̂, sup_x̂) ← ProcessWithDataBubble(Q, T^M).
5: If sup_x̂ ≥ 1/ε^2 then
6:     Output x̂;
7: Else
8:     If the low-frequency group index is usable
9:         x̂ ← ProcessWithLFIndex(Q);
10:    Else
11:        x̂ ← ProcessWithMAIndex(Q);
12:    Output x̂.
```

Algorithm 1: Processing Aggregation Queries

We set s_0 to be $1/\sqrt{n}$, where n is the number of rows in T, in most of discussion and analysis later on. We simply call the two classes large queries and small queries when $s_0 = 1/\sqrt{n}$.

3.1.2 Offline Sampling and Index Building

For a table T with d_{C} group-by and predicate dimensions, d_{M} measures, and n rows, we build three types of samples/indexes.

The first one is called *data bubble*. Each data bubble is a *random sample* of rows drawn from T. We draw a *uniform sample* $T^0 \subseteq T$ containing $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ rows to answer COUNT-queries, and a *measure-biased sample* containing $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ rows T^M used to answer $\mathsf{SUM}(M)$-queries for each measure dimension M. So there are a total of $d_{\mathsf{M}}+1$ data bubbles. The basic idea of measure-biased sampling is to pick a row t in T with replacement with probability proportional to t_M, to get ϵ-approximate answers to $\mathsf{SUM}(M)$-queries using a minimum number of sample rows. As their sizes are sublinear in n, we load them into memory in pre-processing.

The second one is called *measure-augmented inverted index*. For each value of a categorical dimension, we keep a *postings list* of row IDs where this value appears in the corresponding dimension. Up until now, it is similar to an inverted index in IR systems. We then augment each postings list with rough values of measures. For example, if $t_M = 2^8 + 10$, we store 8 (*i.e.*, $\mathsf{apx}(t_M) = 2^8$) for t_M in the index to save space. Similar to measure-biased sampling, these rough values $\mathsf{apx}(t_M)$'s will be used to guide our online sampling when processing SUM-aggregation queries. We keep this index on disk if we do not have enough memory, but each postings list is stored sequentially s.t. it can be efficiently read when needed.

The third one is called *low-frequency group index*. In this index, for each value that is *infrequent* in a categorical dimension, *i.e.*, the number of rows containing it is no more than $\sqrt{n}$, we materialize these rows sequentially on disk. The motivation of this index is that, if an infrequent value appears in the predicate of a query as part of a conjunction, we can simply scan these rows sequentially on disk to calculate the answer. For example, consider the table T in Figure 1(a) and a predicate "$C_2 = 1$ AND $C_3 = 0$". Only 10 rows satisfy $C_2 = 1$, so we can simply scan them for the answer.

3.1.3 Sample+Seek Processing of Queries

Our system does *not* estimate queries' selectivities. For every incoming query, we first proceed with our uniform sample or measure-biased samples. If enough number $(1/\epsilon^2)$ of rows satisfying its predicate are collected from the samples, an ϵ-approximation can be produced. For large queries, the processing can be terminated at this point with high probability. Only when our system does not collect enough number of rows satisfying the predicate from the samples, it continues to use the other two deterministic auxiliary indexes to evaluate the query for ϵ-approximation.

Our query processing algorithm is outlined in Algorithm 1. We process a query with the data bubble samples first (lines 1-4). Let $\hat{\mathbf{x}}$ be the estimated distribution answer, and $\sup_{\hat{\mathbf{x}}}$ be the number of sample rows satisfying the predicate $\mathcal{P}$ (those used to calculate $\hat{\mathbf{x}}$). If $\sup_{\hat{\mathbf{x}}} \geq 1/\epsilon^2$, we can terminate with an ϵ-approximate answer $\hat{\mathbf{x}}$ (lines 5-6); otherwise, $\hat{\mathbf{x}}$ is not accurate enough, so we will continue to check whether the low-frequency group index is usable (lines 8-9), or refer to the measure-augmented inverted index (lines 10-11).

Section 4 introduces how to create and utilize data bubble samples. Section 5 presents how to build and process queries with measure-augmented inverted index and low-frequency group index.

3.1.4 A Summary of Theoretical Guarantees

We now present the performance guarantees of our system for producing ϵ-approximations, in an informal way.

Main results. For a table T with d_{C} group-by/predicate dimensions, d_{M} measures, and n rows. We need to keep $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ rows in each of the $d_{\mathsf{M}}+1$ data bubbles. For any aggregation query $Q(\mathcal{G}, F(M), \mathcal{P})$, we need to scan *one* of the data bubbles to compute ϵ-approximate answer in linear time $\mathrm{O}\big(|\mathcal{G}| \cdot \sqrt{n}/\epsilon^2\big)$. If Q is a large query, we can terminate with an ϵ-approximation with high probability. If Q is a small query and the low-frequency group index can be used to answer it, we scan at most $\sqrt{n}$ rows in the index to get its answer. If the low-frequency group index cannot be used, we can always use the measure-augmented inverted index to perform $\mathrm{O}\big(1/\epsilon^2\big)$ random I/O's for an ϵ-approximation.

Parameters. Both the error bound ϵ and the selectivity threshold s_0 are tunable. Our theoretical results and system can be extended for general parameter settings. The precision parameter ϵ can be specified by users or administrators to trade-off between precision and indexing/processing cost. The selectivity threshold s_0 is set as $1/\sqrt{n}$ by default. Generally, the size of each sample needs to be $\mathrm{O}\big((1/s_0) \cdot (1/\epsilon^2)\big)$ to get ϵ-approximations for s_0-large queries.

Error bound and accuracy probability in big O notations. All the theoretical results about obtaining ϵ-approximation in this paper hold *with high probability* ("*w.h.p.*" for short) – we mean that, with probability at least $1 - \delta$ (*e.g.*, 0.95) for some constant δ (*e.g.*, 0.05), our estimation is an ϵ-approximation. For a fixed error bound ϵ, there is a tradeoff between the sample sizes/processing costs (those in "main results" above) and the accuracy probability $1-\delta$: theoretically, for an accuracy probability $1-\delta$, there is an additional factor $\log(1/\delta)$ behind the $\mathrm{O}(\cdot)$ notations about the sample sizes/processing costs. However, since δ does not have to be arbitrarily small, we can treat $\log(1/\delta)$ as a constant for simplicity of presentation. So in all $\mathrm{O}(\cdot)$ or $\Omega(\cdot)$ notations in the main body of this paper, we omit all the small logarithmic terms like $\log(1/\delta)$ and $\log(1/\epsilon)$ when presenting theoretical results. These terms will be found in the proofs of our theorems in the appendix.

3.2 System Architecture

The architecture of our system is in Figure 2. Our system can be attached to a database with its independent storage and query processing engine. It connects to a data source in the *column-based format* using a component called *data connector*, which can be customized for different database systems. In the index building stage, the three of indexes, *data bubbles*, *measure-augmented inverted index*, and *low-frequency group index*, are created from the column-based storage. After index building is finished, our *query processor* needs to access the data bubbles (in memory) and the other two indexes (on disk) to process aggregation queries.

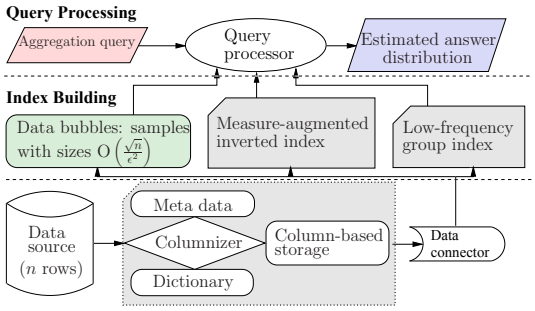


Figure 2: System Architecture

Input: aggregation query $Q(\mathcal{G}, F(M), \mathcal{P})$ and sample T'
Output: ϵ-approximate answer $\hat{\mathbf{x}}$ and support $\sup_{\hat{\mathbf{x}}}$

1: $\hat{\mathbf{x}} \leftarrow \mathbf{0}$; $\sup_{\hat{\mathbf{x}}} \leftarrow 0$;
2: For each sample row $t \in T'$:
3: If t satisfies $\mathcal{P}$ then
4: Let i be the group t belonging to on $\mathcal{G}$;
5: $\hat{x}_i \leftarrow \hat{x}_i + 1$; $\sup_{\hat{\mathbf{x}}} \leftarrow \sup_{\hat{\mathbf{x}}} + 1$;
6: Normalize $\hat{\mathbf{x}}$ to be a distribution on all groups on $\mathcal{G}$;
7: Output $(\hat{\mathbf{x}}, \sup_{\hat{\mathbf{x}}})$.

Algorithm 2: ProcessWithDataBubble(Q, T'): processing aggregation queries with a data bubble (sample)

4. SAMPLE: UNIFORM OR BIASED

We now introduce the two sampling schemes, *uniform sampling* and *measure-biased sampling*, and how to use them to answer large COUNT and SUM-queries with ϵ-approximation, respectively.

4.1 Uniform Sampling

Uniform sampling is a very natural idea to approximate the answer distribution $\mathbf{x}$ for an aggregation query Q. The sampling process can be formally described as follows. For a given sample size m, we repeat drawing a row with replacement from a table T for m times – each row $t \in T$ is chosen into T^0 with equal probability

$$\mathbf{Pr}[t \text{ is picked}] = 1/n. \tag{3}$$

The resulting m sample rows in T^0 are stored in memory for processing queries. We show that a pre-drawn uniform sample with size $\mathrm{O}(\sqrt{n}/\epsilon^2)$ suffices for all large COUNT-queries against T.

Algorithm 2 shows how the uniform sample T^0 can be used to estimate answers to aggregation queries. When $Q(\mathcal{G}, \mathsf{COUNT}(*), \mathcal{P})$ comes, we process it with T^0 using one scan. For each row $t \in T^0$, we check whether t satisfies $\mathcal{P}$ (line 3); if yes and t belongs to the ith group on group-by columns $\mathcal{G}$, we increase $\hat{x}_i$ by 1 (lines 4-5). Finally, we normalize $\hat{\mathbf{x}} = \langle \hat{x}_1, \ldots, \hat{x}_r \rangle$ as a distribution and also report the number of rows satisfying $\mathcal{P}$ as $\sup_{\hat{\mathbf{x}}}$ (lines 6-7).

The follow theorem formally shows that why T^0 with sample size $\mathrm{O}(\sqrt{n}/\epsilon^2)$ suffices for all large COUNT-queries.

THEOREM 3. (Precision of Uniform Sampling) *We use uniform sampling to draw $\mathrm{O}((1/s_0) \cdot (1/\epsilon^2))$ rows to form a sample T^0 from a table T of n rows. For any* COUNT*-aggregation query on T, $Q(\mathcal{G}, \mathsf{COUNT}(*), \mathcal{P})$, let $\mathbf{x}$ be the exact answer to Q. If the selectivity of predicates $\mathcal{P}$ is no less than s_0, $s(\mathcal{P}) \geq s_0$, Algorithm 2 uses T^0 to calculate an estimated answer $\hat{\mathbf{x}}$ s.t. with high probability, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$. In particular, if the selectivity threshold $s_0 = 1/\sqrt{n}$, a uniform sample of $\mathrm{O}(\sqrt{n}/\epsilon^2)$ rows suffices.*

A special case of Theorem 3 with selectivity threshold $s_0 = 1$ has been studied in [1], as stated in Corollary 4 below. Our proof of Theorem 3 generalizes it extensively from two aspects: i) to show $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] \leq \mathrm{O}(\epsilon^2)$, we need to utilize the fact that the number of sample rows satisfying $\mathcal{P}$, denoted as $m_{\mathcal{P}}$, concentrates on $\Omega(1/\epsilon^2)$ (w.h.p.); and ii) we need to generalize McDiarmid's inequality to complete the proof for $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$ (w.h.p.).

COROLLARY 4. (Without Predicate [1]) *We use uniform sampling to draw $\mathrm{O}(1/\epsilon^2)$ rows T^0 from a table T of n rows. For any aggregation query Q with* COUNT *aggregate on T and no predicate, let $\mathbf{x}$ be the exact answer to Q. We can use T^0 to calculate an estimated answer $\hat{\mathbf{x}}$ s.t. with high probability, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \leq \epsilon$.*

Optimality. The information-theoretic lower bound in [1] also implies the optimal sampling size for COUNT, *i.e.*, at least $\Omega(1/\epsilon^2)$ sample rows are needed to obtain an ϵ-approximate answer (with probability at least $1 - \delta$) for a COUNT-query with no predicate.

Early termination. We can derive an early termination condition for Algorithm 2 from Corollary 4. In order to be an ϵ-approximate distribution answer to a query Q, $\hat{\mathbf{x}}$ needs a sample of $\mathrm{O}(1/\epsilon^2)$ rows among all rows that satisfy Q's predicate. So we can track the size of this sample using $\sup_{\hat{\mathbf{x}}}$ in Algorithm 2. When $\sup_{\hat{\mathbf{x}}} \geq 2/\epsilon^2$, we can exit the loop of lines 2-5 and output the normalized $\hat{\mathbf{x}}$. We have integrated this early termination condition in our implementation. It offers significant performance gain for large queries whose selectivities $s(\mathcal{P})$'s are much larger than $1/\sqrt{n}$, as for those queries, only a small portion of rows in the $\mathrm{O}(\sqrt{n}/\epsilon^2)$-sample need to be scanned to get a sample of $2/\epsilon^2$ rows satisfying $\mathcal{P}$.

4.2 Measure-biased Sampling

To extend the uniform sampling schema for queries with SUM aggregates on any measure dimension M, a naive way is to change "$\hat{x}_i \leftarrow \hat{x}_i + 1$" to "$\hat{x}_i \leftarrow \hat{x}_i + t_M$" on line 5 of Algorithm 2 (recall that t_M is the value of measure M on a row t). This simple extension, however, is not accurate enough. We can show that using a uniform sample with size $m = \mathrm{O}(\sqrt{n}/\epsilon^2)$, the expected squared error $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right]$ for large queries is *no less* than $\mathbf{Var}[M] \cdot \epsilon^2$, where $\mathbf{Var}[M] = \frac{1}{n}\sum_{t \in T}(t_M/\bar{M} - 1)^2$ and $\bar{M}$ is the average value of measure M. Later in Proposition 7, we will also prove that $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right]$ is upper bounded by $\mathrm{O}(\Delta^3 \cdot \epsilon^2)$ with the same sample size, if measure M takes value in $[1, \Delta]$. Both $\mathbf{Var}[M]$ and Δ could be very large for a general data distribution on M.

Recall cases I and II in Example 1.1. If we use the above naive adaption of uniform sampling to process query Q, we will get an estimated answer either in Figure 1(c) or in 1(d) with high probability. Both are far away from the true answer.

We now introduce a new sampling schema called *measure-biased sampling*. For each measure M, we first need to construct (randomly) a *measure-biased sample* T^M in preprocessing, which will be used to estimate answers for SUM-queries with high accuracy.

Constructing measure-biased samples. To construct T^M with a given sample size m, we repeat drawing a row with replacement from T for m times – each row t is picked into T^M with probability proportional to its value t_M on measure M, *i.e.*,

$$\mathbf{Pr}[t \text{ is picked}] = \frac{t_M}{\sum_{t' \in T} t'_M} \sim t_M; \tag{4}$$

All the rows drawn are kept in T^M. Note that each row may appear multiple times in T^M, *i.e.*, T^M could be a multi-set. We will show that for a table T with n rows, a measure-biased sample T^M with size $\mathrm{O}(\sqrt{n}/\epsilon^2)$ suffices for all large SUM(M)-queries.

If there are d measure attributes, we need to maintain d measure-biased samples, each for one measure. We can construct all the

measure-biased samples together with the uniform sample in two scans of the table T. Please refer to Appendix B for details.

Estimating using measure-biased samples. When using T^M to estimate the answer for a $\mathsf{SUM}(M)$-query Q, it is a bit surprising that we can reuse Algorithm 2 – we only need to call the procedure $\mathsf{ProcessWithDataBubble}(Q, T^M)$. Readers may wonder why in line 5, $\hat{x}_i$ is increased by 1 instead of the measure value t_M for a sample row t. The reason is, informally, that since the sampling probability of t is proportional to t_M (as in (4)), we need to calibrate row t's contribution to the estimation $\hat{\mathbf{x}}$ with t_M^{-1}. This idea is similar to Horvitz-Thompson estimator (which is for single-point estimation), but here we are estimating a distribution $\mathbf{x}$. Refer to the following example for more intuition.

EXAMPLE 4.1. (Continue Example 1.1) *Let's look at the query Q in Example 1.1, for which uniform sample fails. We draw a measure-biased sample T^M with rows 1, 26, 51, 76, 92, 94, 96, 98, 111, 136, 161, and 186 each appearing once, and 199 and 200 each appearing four times (considering their values on M – refer to Example B.1 and Figure 9 in Appendix B on how they were drawn). Reusing Algorithm 2, call $\mathsf{ProcessWithDataBubble}(Q, T^M)$ – the estimation it gives is $\hat{\mathbf{x}} = \langle 8/20, 12/20\rangle = \langle 0.40, 0.60\rangle$, which is very close to the exact answer $\mathbf{x} = \langle 0.39, 0.61\rangle$.*

The following theorem formally shows that, for each measure M, a measure-biased sample T^M with size $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ suffices for all large $\mathsf{SUM}(M)$-queries to give ϵ-approximation.

THEOREM 5. (Precision of Measure-Biased Sampling) *We apply measure-biased sampling on measure attribute M and draw $\mathrm{O}\big((1/s_0)\cdot(1/\epsilon^2)\big)$ rows T^M from a table T of n rows. For any aggregation query, $Q(\mathcal{G}, \mathsf{SUM}(M), \mathcal{P})$, with aggregate $\mathsf{SUM}(M)$ on T, let $\mathbf{x}$ be the exact answer to Q. If the measure selectivity of predicates $\mathcal{P}$, $s_M(\mathcal{P})$, is no less than s_0, Algorithm 2 uses T^M to calculate an estimated answer $\hat{\mathbf{x}}$ s.t. with high probability, $\|\mathbf{x}-\hat{\mathbf{x}}\|_2 \le \epsilon$. In particular, if the selectivity threshold $s_0 = 1/\sqrt{n}$, a measure-biased sample of $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ rows suffices.*

Optimality of measure-biased sampling. Our measure-biased sampling for SUM-queries draws the same number of sample rows as the uniform sampling does for COUNT-queries. Since COUNT is a special case of SUM (when the measure attribute is a constant 1), the low-bound result in [1] also implies that for SUM-queries with no predicate ($s_0 = 1$), we need at least $\Omega\big(1/\epsilon^2\big)$ sample rows to obtain an ϵ-approximate answer with high probability.

Early termination. The same early termination condition introduced in Section 4.1 for the uniform samples can be applied here for measure-biased samples. When we collect $\sup_{\hat{\mathbf{x}}} \ge 2/\epsilon^2$ samples rows that satisfy the predicate in Q, we can exit the loop of lines 2-5 of Algorithm 2 and output the normalized $\hat{\mathbf{x}}$. This condition is also integrated in our implementation and offers significant performance gain for queries with high measure selectivities.

5. SEEK: RANDOM OR SEQUENTIAL

Large queries with (measure) selectivity no smaller than s_0 ($= 1/\sqrt{n}$ by default) have been handled by uniform and measure-biased samples. So what left here are small queries with (measure) selectivity smaller than s_0. For example, for such a small COUNT aggregation query, we know that the number of rows satisfying its predicate is no more than $\sqrt{n}$. We will keep this property in mind when designing the two indexes to handle small queries.

5.1 Measure-Augmented Inverted Index

As we first focus on aggregation queries with categorical dimensions in their predicates, it is a natural idea to apply *inverted indexes* from IR systems in our problem. For each value v of each dimension D_i, we can keep a *postings list* of rows that have value v on dimension D_i. Then for a query $Q(\mathcal{G}, F(M), \mathcal{P})$, there have been extensive studies on how to efficiently retrieve all rows satisfying $\mathcal{P}$ using those postings lists, such as [8]. For a predicate that is a conjunction (with only ANDs) of equi-constraints, it is equivalent to computing the intersection of postings lists. After that, we only need to scan the retrieved rows once for aggregation.

There are, however, at least two optimizations we can do in the scenario of aggregation queries processing. First, we cannot keep full rows in the postings list, as each row could be wide (in terms of the number of bits), especially if the number of dimensions is non-trivial (*e.g.*, 30). We can keep only the IDs of those rows, and later seek their dimension/measure values in the original table.

Secondly, of course, random seek based on row ID is expensive. Our basic idea is that, after we get the IDs of rows that satisfy the predicate $\mathcal{P}$, we draw a random sample from these IDs, and perform random seeks only for the sample IDs. The two sampling schemes introduced in Section 4 can be applied here online to draw proper samples to achieve our ϵ-approximation precision guarantee. To this end, besides row IDs, we need approximate measure values to guide our online measure-biased sampling.

We introduce how to construct and utilize *measure-augmented inverted index* in this subsection.

5.1.1 Construction

Our measure-augmented inverted index is described in Figure 3. For each value v of a categorical dimension D, we maintain a *measure-augmented postings list* $\mathsf{inv}(D, v)$. Each of our measure-augmented postings lists consists of two parts.

First, $\mathsf{inv}(D, v)$ maintains IDs of rows that have value v on D just as in an IR-style postings list. This part is depicted in the left of Figure 3. From the ID of a row t, denoted as t_{ID}, we can access the value of each the dimension or measure of t from our column-based storage, using one random seek (I/O).

Second, let's use $\mathsf{inv}(D, v)$ also to denote the set of rows that have value v on D. For each row $t \in \mathsf{inv}(D, v)$, we maintain the values of measures $M_1, \ldots, M_{d_{\mathsf{M}}}$ on this row. Note that keeping all the exact measure values is still too expensive (*e.g.*, for ten 64-bit integers). So we only keep an approximation $\mathsf{apx}(t_{M_i})$ of measure t_{M_i} in order to save the space as well as to speedup the access of $\mathsf{inv}(D, v)$. $\mathsf{apx}(t_{M_i})$ only needs to be an 2-*approximation* of t_{M_i}:

$$\mathsf{apx}(t_{M_i}) \le t_{M_i} < 2 \cdot \mathsf{apx}(t_{M_i}). \tag{5}$$

So the number of bits we need is at most $\log\log\Delta(M_i)+1$, where $\Delta(M_i)$ is the range of values in measure M_i.

The construction of measure-augmented inverted index is similar to constructing traditional IR inverted index, which can be done within one scan of all the rows in linear time.

EXAMPLE 5.1. (Continue Example 1.1) *For the example table T with 200 rows in Figure 1(a), Figure 4(a)-4(b) shows the measure-augmented postings lists $\mathsf{inv}(C_2, 0)$ and $\mathsf{inv}(C_3, 0)$, respectively. Values on measure M,* i.e., *t_M, are rounded to $\mathsf{apx}(t_M)$ as $2^0, 2^1, 2^2, \ldots$, so that (5) is satisfied. For example, t_M for row 100 is rounded to 8 from 10 in $\mathsf{inv}(C_3, 0)$ and t_M for row 200 is rounded to 64 from 100 in $\mathsf{inv}(C_2, 0)$. The number of bits needed for each $\mathsf{apx}(t_M)$ is $\log_2\log_2\Delta(M)+1$. In this example, the range of M is $\Delta(M) = 100$, so we need only $\log_2\log_2 100 + 1 = \log_2 6 + 1 = 3$ bits, as $\mathsf{apx}(t_M)$ can only be $2^0, 2^1, \ldots, 2^6$.*

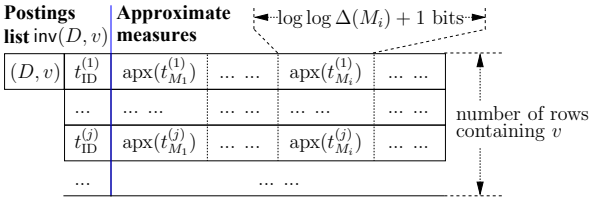


Figure 3: Measure-Augmented Inverted Index

ID	$\mathsf{apx}(M)$
1	1
…	1
90	1
101	1
…	1
198	1
199	64
200	64

(a) $\mathsf{inv}(C_2, 0)$

ID	$\mathsf{apx}(M)$
1	1
…	1
90	1
91	8
…	8
100	8
101	1
…	1
198	1
199	64

(b) $\mathsf{inv}(C_3, 0)$

ID	$\mathsf{apx}(M)$
1	1
…	1
90	1
101	1
…	1
198	1
199	64

(c) $\mathsf{inv}(C_2, 0) \cap \mathsf{inv}(C_3, 0)$

Figure 4: Two Measure-Augmented Postings Lists

Size of index. Our measure-augmented inverted index is composed of row IDs and approximate measures. Suppose a table has n rows and d_{C} dimensions. Each row ID needs $\mathrm{O}(\log n)$ bits and is stored d_{C} times in the index, one time for each dimension. For each row ID and each measure M_i, we need $\mathrm{O}(\log\log\Delta(M_i))$ bits to store the approximate measure $\mathsf{apx}(t_{M_i})$. So the index size is:

THEOREM 6. *For a table with n rows, d_{C} dimensions, and d_{M} measures, the measure-augmented inverted index needs a total of $\mathrm{O}\Big(n \cdot d_{\mathsf{C}} \cdot (\log n + \Sigma_{i=1}^{d_{\mathsf{M}}} \log\log\Delta(M_i))\Big)$ bits.*

Each postings list $\mathsf{inv}(D, v)$ can be compressed using standard techniques as it will only be accessed sequentially. We will report the actual size of such indexes in our experiments.

5.1.2 Processing Queries

To process a query $Q(\mathcal{G}, F(M), \mathcal{P})$ using measure-augmented inverted index, we first retrieve IDs of all rows that satisfy $\mathcal{P}$. Let $T_{\mathcal{P}}$ be the set of all such rows, and $T^{\mathsf{ID}}_{\mathcal{P}}$ be the set of their IDs. For a predicate expression with AND, OR, and NOT operators on equi-constraints in the form of "$D_i = v_i$", this is equivalent to computing set intersection, union, and minus of postings lists $\mathsf{inv}(D_i, v_i)$, and we only need one scan of those lists.

After we get IDs of rows in $T^{\mathsf{ID}}_{\mathcal{P}}$, we can seek the original table T for their dimension and measure values to finish the aggregation. But $|T^{\mathsf{ID}}_{\mathcal{P}}|$ random seeks are expensive.

A very natural idea is to draw a random sample from $T^{\mathsf{ID}}_{\mathcal{P}}$, and perform random seeks only for the sample rows.

For COUNT aggregates, from Corollary 4, a uniform sample of $\mathrm{O}\big(1/\epsilon^2\big)$ rows suffice for an ϵ-approximate answer $\hat{\mathbf{x}}$.

For SUM aggregates, we propose *approximate measure-biased sampling* to estimate an ϵ-approximate answer.

Online approximate measure-biased sampling. We start with a proposition about the sample size needed for an ϵ-approximation if *uniform sampling* is used. We prove that, with uniform sampling, it suffices to seek $\mathrm{O}\big(\Delta^3/\epsilon^2\big)$ sample rows, where Δ is the range of measure values. This result itself is discouraging as Δ^3 is large. Our *approximate measure-biased sampling* scheme will improve the sample size to $\mathrm{O}\big(1/\epsilon^2\big)$ using approximate measures $\mathsf{apx}(\cdot_M)$.

PROPOSITION 7. (Uniform Sample for SUM) *We draw a uniform sample of $\mathrm{O}\big(\Delta^3/\epsilon^2\big)$ rows $T'_{\mathcal{P}}$ from $T_{\mathcal{P}}$, assuming values of measure M are within range $[1, \Delta]$. For a* SUM *aggregation query Q with predicate $\mathcal{P}$, let $\mathbf{x}$ be the exact answer to Q. We can use $T'_{\mathcal{P}}$ to get an estimation $\hat{\mathbf{x}}$ s.t. $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2\right] \le \epsilon$.*

Input: aggregation query $Q(\mathcal{G}, F(M), \mathcal{P})$ and table T
Index: measure-augmented inverted index inv
Output: ϵ-approximate answer distribution $\hat{\mathbf{x}}$
1: $\hat{\mathbf{x}} \leftarrow \mathbf{0}$;
2: Compute $T^{\mathsf{ID}}_{\mathcal{P}} \leftarrow \{t_{\mathsf{ID}} \mid$ rows t that satisfy $\mathcal{P}\}$ using inv;
3: Repeat $\min\{1/\epsilon^2, |T^{\mathsf{ID}}_{\mathcal{P}}|\}$ times:
4: Randomly pick $t_{\mathsf{ID}} \in T^{\mathsf{ID}}_{\mathcal{P}}$: for each $t_{\mathsf{ID}} \in T^{\mathsf{ID}}_{\mathcal{P}}$,

$$\mathbf{Pr}\left[t_{\mathsf{ID}} \text{ is picked}\right] = \frac{\mathsf{apx}(t_M)}{\sum_{t' \in T} \mathsf{apx}(t'_M)} \sim \mathsf{apx}(t_M); \quad (6)$$

5: $t \leftarrow$ seeking T using t_{ID} for other dimensions;
6: Let i be the group t belonging to on $\mathcal{G}$;
7: $\hat{x}_i \leftarrow \hat{x}_i + t_M/\mathsf{apx}(t_M)$;
8: Normalize $\hat{\mathbf{x}}$ to be a distribution and output $\hat{\mathbf{x}}$.

Algorithm 3: ProcessWithMAIndex(Q): online approximate measure-biased sampling and processing

How to process a query using the approximate measure-biased sampling technique is described in Algorithm 3. Inspired by the idea of measure-biased sampling (recall (4)), the sample size can be reduced tremendously to $\mathrm{O}\big(1/\epsilon^2\big)$ if rows in $T_{\mathcal{P}}$ can be sampled with probability proportional to t_M. In measure-augmented inverted index, t_M is not available, but an approximation to t_M, $\mathsf{apx}(t_M)$, is available. A row t in $T_{\mathcal{P}}$ is drawn into sample $T'_{\mathcal{P}}$ with probability proportional to $\mathsf{apx}(t_M)$ (line 4). If t is picked, we retrieve its dimension and measure values by seeking our column-based storage (line 5), and let i be the group it belongs to (lines 6). Let $t_{M'} = t_M/\mathsf{apx}(t_M)$ be the weight of t in the estimation $\hat{x}_i$ (line 7). Finally, $\hat{\mathbf{x}}$ is normalized and output (line 8). We know that, for any t, $1 \le t_{M'} < 2$, from how $\mathsf{apx}(\cdot_M)$ is chosen as in (5). The reduction in sampling complexity is achieved by shrinking Δ in Proposition 7. Intuitively, from Proposition 7, if $\Delta = 2$, which is true for $t_{M'}$, a sample of size $\mathrm{O}\big(1/\epsilon^2\big)$ should suffice. Before proving the formal result in Theorem 8, we use an example to demonstrate the online approximate measure-biased sampling.

EXAMPLE 5.2. (Continue Example 5.1) *For the table T with 200 rows in Figure 1(a), consider such a query Q:*

SELECT C_1, SUM(M) FROM T
WHERE $C_2 = 0$ AND $C_3 = 0$ GROUP BY C_1.

The answer $\mathbf{x} = \langle 90/288, 198/288\rangle = \langle 0.31, 0.69\rangle$. Using our measure-augmented postings lists in Figures 4(a)-4(b), we can first compute the IDs of rows that satisfy "$C_2 = 0$ AND $C_3 = 0$". We get $T^{\mathsf{ID}}_{\mathcal{P}} = \{1, \ldots, 90, 101, \ldots, 198, 199\}$ as in Figure 4(c). There are 188 rows in $T^{\mathsf{ID}}_{\mathcal{P}}$ with $\mathsf{apx}(t_M) = 1$, and 1 row (199) with $\mathsf{apx}(t_M) = 64$. So if we draw 20 rows with probability proportional to $\mathsf{apx}(t_M)$ from $T^{\mathsf{ID}}_{\mathcal{P}}$, with high likelihood, we will get 7 rows from $\{1, \ldots, 90\}$, 8 rows from $\{101, \ldots, 198\}$, 5 copies of row 199. From line 7 in Algorithm 3, we can estimate $\hat{\mathbf{x}} =$

$$\langle \frac{7 \times 1}{7 + 8 + 5 \times 100/64}, \frac{8 \times 1 + 5 \times 100/64}{7 + 8 + 5 \times 100/64} \rangle = \langle 0.30, 0.70\rangle.$$

THEOREM 8. (Sample using Approximate Measures) *We can draw $\mathrm{O}\big(1/\epsilon^2\big)$ rows $T'_{\mathcal{P}}$ from $T_{\mathcal{P}}$ biased on approximate measure $\mathsf{apx}(\cdot_M)$ of M (in (6)). For an aggregation query $Q(\mathcal{G},$ SUM(M), $\mathcal{P})$, let $\mathbf{x}$ be the exact answer to Q. We can use the measure-biased sample $T'_{\mathcal{P}}$ to calculate an approximate answer $\hat{\mathbf{x}}$ (as in Algorithm 3) s.t. with high probability, we have $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \le \epsilon$.*

Query processing cost. We observe the bottleneck in cost lies on the second step, because the first step only involves sequential scans of postings lists, but the second step needs random seeks. The number of random seeks, however, is bounded by $\min\{1/\epsilon^2, |T_{\mathcal{P}}^{\mathsf{ID}}|\}$ for each dimension in the query to get an ϵ-approximation.

THEOREM 9. *A query Q with predicate $\mathcal{P}$ can be processed using our measure-augmented inverted index (with Algorithm 3) to get an ϵ-approximation. Let $T_{\mathcal{P}}$ be the set of rows satisfying $\mathcal{P}$. The total number of random seeks to our index and the column-based storage is $\mathrm{O}\big(\min\{1/\epsilon^2, |T_{\mathcal{P}}|\} \cdot d_Q\big)$, where d_Q is the number of dimensions and measures in the query Q.*

5.2 Low-Frequency Group Index

Low-frequency group index is designed to benefit a special class of predicates in the form of

$$\mathcal{P} : D_1 = v_1 \text{ AND } \ldots \text{ AND } D_l = v_l \text{ AND } (\ldots), \qquad (7)$$

i.e., a conjunction of one or more equi-constraints and some other constraints. For example, for table T in Figure 1(a), "$C_2 = 1$ AND $(C_3 = 0$ OR $C_3 = 1)$" is such a predicate.

The construction and processing using low-frequency group index is quite straightforward. If there is less than $\sqrt{n}$ rows (out of all the n rows in the table T) with value v on a dimension D, we simply store these rows sequentially in the index. Any query in the form (7) with "$D = v$" in its predicate is a small query and cannot be handled by our samples. Processing such queries using the low-frequency group index is easy: we only need to scan the $\sqrt{n}$ rows sequentially in the index. More details are in Appendix C.

Not all small queries can be processed using this index. For example, when $T_{D_1=v_1}$ and $T_{D_2=v_2}$ are both large (with $> \sqrt{n}$ rows and thus not in the index), their intersection $T_{D_1=v_1} \cap T_{D_2=v_2}$ could be small. We still rely on our measure-augmented inverted index to answer the query with predicate "$D_1 = v_1$ AND $D_2 = v_2$".

6. EXTENSIONS

6.1 From Distributions to Absolute Answers

We now introduce how to convert a distribution answer $\hat{\mathbf{x}}$ to an absolute answer $\hat{\mathbf{z}}$. For the distribution answer obtained from low-frequency group index, since it is exact, the conversion is trivial. So let's focus on SUM-queries with estimated distribution obtained from measure-biased sampling. Measure-augmented inverted index also relies on (approximate) measure-biased sampling, so it is similar. And COUNT is a special case of SUM.

Consider a $\mathsf{SUM}(M)$-query Q with predicate $\mathcal{P}$ against a table T. Suppose its output is $\mathbf{z} = \langle z_1, \ldots, z_r \rangle$, where z_i is the aggregate value of the ith group. $\mathbf{z}$ can be normalized to a distribution $\mathbf{x} = \langle x_1, \ldots, x_r \rangle$ as in Section 2.1. Let $T_{\mathcal{P}} \subseteq T$ be the set of rows satisfying $\mathcal{P}$ and $T_i \subseteq T_{\mathcal{P}}$ be the set of rows in the ith group of the output to Q. For a set of rows S, let $M(S) = \sum_{t \in S} t_M$. So we can easily derive that $x_i = M(T_i)/M(T_{\mathcal{P}}) = z_i/M(T_{\mathcal{P}})$.

In the measure-biased sample T^M, let m be the number of sample rows in T^M, $m_{\mathcal{P}}$ be the number of sample rows satisfying $\mathcal{P}$ in T^M, and m_i be the be the number of rows belonging to the ith group in T^M. So from Algorithm 2, we have $\hat{x}_i = m_i/m_{\mathcal{P}}$.

Since $|\hat{x}_i - x_i| \le \epsilon$ (implied by our distribution precision guarantee $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \le \epsilon$), informally, we have

$$x_i = z_i/M(T_{\mathcal{P}}) \approx m_i/m_{\mathcal{P}} = \hat{x}_i.$$

So it is intuitive to estimate $\hat{z}_i = M(T_{\mathcal{P}}) \cdot m_i/m_{\mathcal{P}}$. Both m_i and $m_{\mathcal{P}}$ are known from Algorithm 2, but $M(T_{\mathcal{P}})$ is unknown. Alternatively, we can estimate $M(T_{\mathcal{P}})$ as $M(T) \cdot m_{\mathcal{P}}/m$, and therefore,

$$\hat{z}_i = M(T) \cdot m_i/m = \hat{x}_i \cdot M(T) \cdot m_{\mathcal{P}}/m.$$

Negative values on measures. For a measure attribute M with both positive values and negative values, we can create two versions of M: M^+ takes only positive values and is set to 0 if the row has a negative value on M; and similarly, M^- takes only negative values and is set to 0 if the row has a positive value on M. Using the above tricks, we can get estimated answers for both M^+ and M^-. The estimation for M can be obtained from the difference between the estimated M^+ and M^- on each group. In this way, we can at least handle negative values for both SUM and AVG.

Input: aggregation query $Q(\mathcal{G}, F(M), \mathcal{P})$ and sample T'

$\hat{S}_i \leftarrow \varnothing$ for all groups i's; $\sup \leftarrow 0$;
For each sample row $t \in T'$:
  If t satisfies $\mathcal{P}$ then
    Let i be the group t belonging to on $\mathcal{G}$;
    $\hat{S}_i \leftarrow \hat{S}_i \cup \{t\}$; $\mathsf{w}(t) \leftarrow \mathbf{Pr}[t \in T']$; $\sup \leftarrow \sup + 1$;
For each group i: $\hat{\mathbf{x}}_i \leftarrow \mathsf{Estimator}(\hat{S}_i, \mathsf{w}(\cdot), F)$;
Output $(\hat{\mathbf{x}}, \sup)$.

Algorithm 4: $\mathsf{ProcessWithDataBubble}(Q, T')$: extended for general measure F on uniform/measure-based sample

Input: aggregation query $Q(\mathcal{G}, F(M), \mathcal{P})$ and table T
Index: measure-augmented index MAInd

$\hat{S}_i \leftarrow \varnothing$ for all groups i's;
Compute $T_{\mathcal{P}}^{\mathsf{ID}} \leftarrow \{t_{\mathsf{ID}} \mid \text{rows } t \text{ that satisfy } \mathcal{P}\}$ using MAInd;
Repeat $\min\{1/\epsilon^2, |T_{\mathcal{P}}^{\mathsf{ID}}|\}$ times:
  Randomly pick $t_{\mathsf{ID}} \in T_{\mathcal{P}}^{\mathsf{ID}}$: for each $t_{\mathsf{ID}} \in T_{\mathcal{P}}^{\mathsf{ID}}$,
    If MeasureBiased $=$ true: $\mathbf{Pr}[t_{\mathsf{ID}}] \sim \mathsf{apx}(t_M)$;
    Else (uniform sampling): $\mathbf{Pr}[t_{\mathsf{ID}}] = 1/|T_{\mathcal{P}}^{\mathsf{ID}}|$;
  $t \leftarrow$ seeking T using t_{ID} for other dimensions;
  Let i be the group t belonging to on $\mathcal{G}$;
  $\hat{S}_i \leftarrow \hat{S}_i \cup \{t\}$; $\mathsf{w}(t) \leftarrow \mathbf{Pr}[t_{\mathsf{ID}}]$;
For each group i: $\hat{\mathbf{x}}_i \leftarrow \mathsf{Estimator}(\hat{S}_i, \mathsf{w}(\cdot), F)$;
Output $\hat{\mathbf{x}}$.

Algorithm 5: $\mathsf{ProcessWithMAIndex}(Q, \mathrm{MeasureBiased})$: extended for general measure F on uniform/measure-based sample

6.2 Extensibility of Our Techniques

Foreign key joins and star schema. In order for our system to support aggregate queries with foreign key joins in a star schema, our samples and indexes can be extended as follows.

Uniform/measure-biased samples are drawn from the fact table in the same way as introduced in Section 4. But for each dimension table, we have two options. i) We can join it with the samples from the fact table in the offline sampling stage and attach the additional dimensions to the samples; or ii) we can perform the join when processing aggregation queries using samples, if one or more of its dimensions appear in the query. Option ii) is feasible when the dimension table is small enough to be fit into the memory. In our implementation, we choose option i). For the complex schema in TPC-H, Table 1 shows the space overhead of all uniform and measure-biased samples, which is affordable in memory even though samples are attached with dimensions from all tables.

For the measure-augmented inverted index and the low-frequency group index, we construct them in the view that joins the fact table with all the dimensions. For TPC-H, Table 1 shows their total sizes.

In a snowflake schema with multiple fact tables and multi-level dimensions, the above extensions can still be applied.

Range predicates. Our samples can already be used to handle any predicate that is based on existing dimensions. To handle range predicates like "$D \geq s$ AND $D \leq t$" for small queries, we introduce the following *measure-augmented B+ tree index*.

• *Measure-augmented B+ tree.* In a B+ tree built on a numerical dimension D, each leaf node is associated with a list of D's values within certain range. In our measure-augmented B+ tree, for each D's value at each leaf, we associate it with IDs (pointers) of the rows with this value on D and also approximate measure $\mathsf{apx}(t_M)$ for each measure (as in (5)) of these rows – recall that these approximate measures need much less bits than the exact measure values.

When a range constraint $s \leq D \leq t$ is in the predicate, we can use this measure-augmented B+ tree to retrieve IDs of rows with $s \leq t_D \leq t$ as well as their approximate measures. Our approximate measure-biased sampling introduced in Section 5.1.2 can then be used to pick $\mathrm{O}\big(1/\epsilon^2\big)$ row IDs and seek their dimension values in the original table to answer a query with ϵ-approximation.

Extension for other aggregates. We now focus on how to extend our sample+seek solution to a wide range of aggregates.

• *Simple extensions.* We can convert both estimated distributions for SUM and COUNT to absolute answers using the way introduced in Section 6.1. Then some measure like AVG can be easily supported as it is nothing but the ratio of SUM to COUNT.

• *Generic aggregates.* To support a generic range of aggregates in Algorithm 1, we extend processing with uniform/measure-biased samples (Algorithm 2) and processing with measure-augmented index (Algorithm 3) to Algorithm 4 and Algorithm 5, respectively. Processing with low-frequency group index is trivial: one can get the exact value of any aggregate by scanning rows in the index.

The main idea behind Algorithms 4-5 is that, no matter with i) uniform/measure-biased samples or ii) with measure-augmented indexes, our estimations are made based on a set of sample rows with different sampling probabilities. For each group in the answer, with i), sample rows have been pre-drawn and kept in memory, so Algorithm 2 can directly use them. With ii), Algorithm 3 does online sampling/seeking in lines 4-5, and make use of the sample rows for estimation in lines 6-7. So it is intuitive to replace the specific estimation methods in Algorithm 2 (lines 4-5) and Algorithm 3 (lines 6-7) with generic estimators to handle other aggregates.

In Algorithm 4, when scanning rows in a (uniform or measure-biased) sample T', we keep rows from each group i in $\hat{S}_i$ as well as the probability of seeing each row t in $\mathsf{w}(t)$ (line 5). After the scan, for each group i, a point estimation of its aggregate F can be made based on the set of sample rows $\hat{S}_i$ from group i and their weights $\mathsf{w}(\cdot)$, using a generic estimator $\mathsf{Estimator}(\hat{S}_i, \mathsf{w}(\cdot), F)$ (line 6).

The extension in Algorithm 5 is similar, but the sampling is conducted online, either uniformly (line 6) or biased on measure (line 5). Again, rows collected from sampling are kept in $\hat{S}_i$ for group i together with sampling probabilities in $\mathsf{w}(\cdot)$ (line 9). After enough number of rows are collected, estimation of aggregate value F for each group can be made using $\mathsf{Estimator}(\hat{S}_i, \mathsf{w}(\cdot), F)$.

Estimator is a customizable component and *any sample-based point estimator* for the aggregate F can be applied. It takes a set of sample rows $\hat{S}$, their sampling weights $w(\cdot)$, and F as the input, but does not rely on other components of our system. So our solution enjoys the same extensibility as other sampling-based AQP techniques (*e.g.* [7, 5]) do to support a generic range of aggregates. For example, if F is $\mathsf{STDEV}(M)$, the sample standard deviation estimator can be used as Estimator; if F is $\mathsf{COUNT}(\mathsf{DISTINCT}\ M)$, the estimator GEE proposed in [10] can be used as Estimator.

We will evaluate such extensions in experiments.

7. EVALUATION

We compare our system with a commercial RDB and two state of the art AQP solutions with stratified sampling: Babcock *et al.* [7] (workload-independent) and BlinkDB [5] (workload-aware).

• SPS: Our *sample+seek* solution introduced in this paper.

• DBX: A standard commercial database system with column-store indexes built for all tables on all columns.[1]

• SMG: Small group sampling introduced in [7].

• BLK: BlinkDB with multi-dim stratified sampling [5].

More implementation details of SPS.

• *Parameters.* Our SPS has two parameters. The selectivity threshold s_0 is set to $1/\sqrt{n}$, where n is the total number of rows in a table. The requested error bound ϵ varies from 0.025 to 0.1, and is set as 0.05 if not specified. We use constants 1 behind all $\mathrm{O}(\cdot)$'s in our theoretical results. For uniform/measure-biased sampling, our analysis shows that $\mathrm{O}\big(\sqrt{n}/\epsilon^2\big)$ rows in each sample suffice – we pick exactly $\sqrt{n}/\epsilon^2$ rows for each sample. Similarly, from Theorem 8, we pick exactly $1/\epsilon^2$ rows to perform random seeks.

• *Independent storage.* We implement our storage/index and query execution engine in SPS using C# from sketch. Our custom implementation (of all the components in Figure 2 except "data source") does not reply on other database systems. In our system architecture in Figure 2, we have the original data kept in a column-based storage with dictionary compression on disk. Our samples/indexes are constructed from it, with samples loaded into memory and the other two types of indexes maintained on disk. DBX has a better optimized column-based storage engine – we do not utilize it for SPS, as we want to highlight the speedups brought by our samples/indexes and processing algorithms with early termination.

• *Column-store v.s. row-store.* Our sample+seek framework is independent on the actual data storage of tables, but we choose column-store for uniform/measure-biased samples, and row-store for low-frequency group indexes based on the ways they are accessed. For in-memory samples, a query may touch only a few dimensions of sample rows and only data on these dimensions need to be read. So to take the best advantage of data locality and cache, it is more efficient to store the data column by column. For on-disk low-frequency group indexes, since at most $\sqrt{n}$ rows need to be scanned to answer a query, it is more efficient to store the data row by row so that only one random access is required per query. Dictionary compression is applied on both types of storages.

SMG and BLK are implemented on the same storage engine (to store data and samples) as SPS, and all the samples from SMG and BLK are pre-loaded into memory for fair comparison.

7.1 Experiment Settings and Summary

All of our experiments are conducted on a Windows server with 8 core 2.27GHz CPUs and 64 GB memory. DBX is given 4 cores for better performance, and each of the rest uses one core.

Datasets. We use two datasets (a real one and a benchmark):

- LOG: This is a real dataset about service logs in a Fortune 500 company. It is a single table with 29 columns – 19 of them are categorical dimensions (for group-by), and the rest 10 are numerical dimensions/measures. The numbers of distinct values in the categorical dimensions range from 2 to 3000. The whole table has 1.12 billion rows. To test the scalability of different systems, we extract two sub-tables from LOG, LOG-S with 310 million rows and LOG-M with 620 million rows.
- TPC-H: With a scaling factor 100, we produce a 100 GB star-

[1]We tried different types of indexes. Column-store indexes turn out to be the best option for our aggregation query workloads.

Datasets	# of rows (n)	Size on disk	On-disk indexes				In-memory uniform and measure-biased samples (only needed for one ϵ)								time
			MA index		LF index		$\epsilon = 0.025$		$\epsilon = 0.050$		$\epsilon = 0.075$		$\epsilon = 0.010$		
			size	time	size	time	budget	size	budget	size	budget	size	budget	size	
LOG-S	310M	60GB	26GB	0.2h	36GB	0.3h	0.0909	11GB	**0.0227**	2.8GB	0.0101	1.2GB	0.0056	0.68GB	0.2h
LOG-M	620M	120GB	41GB	0.5h	6GB	0.4h	0.0643	16GB	**0.0161**	4.1GB	0.0071	1.8Gb	0.0040	1.1GB	0.4h
LOG	1120M	202GB	70GB	1.0h	8GB	0.7h	0.0478	22GB	**0.0120**	5.5GB	0.0053	2.4GB	0.0030	1.4GB	0.7h
TPC-H	600M	100GB	62GB	0.6h	6GB	0.4h	0.0653	15GB	**0.0163**	3.8GB	0.0073	1.7GB	0.0041	0.94GB	0.5h

Table 1: Data size and error bound v.s. sampling budget, sample sizes, and index sizes

schema database using TPC-H benchmark, in which the fact table LINEITEM has a total of 600 million rows.

Workloads. For both datasets, we randomly generate workloads of aggregation queries with predicates, group-bys, and aggregates, and foreign key joins (in TPC-H). We vary the number of group-by columns from 1 to 4 and the number of dimensions in predicates from 0 to 4. COUNT/SUM aggregates are used for experiments in Sections 7.3-7.4. SUM and COUNT(DISTINCT $\cdot$) are tested in Section 7.5. In each aggregation query, the measure M involved is chosen randomly from all the measures. Group-by dimensions and predicate dimensions are chosen uniformly at random from all the categorical dimensions in all tables, with key/id columns excluded (as they are not likely to appear in aggregation queries). In Sections 7.3-7.4, a predicate consists of 0 to 4 (dimension, value) pairs, which represent conjunction (ANDs) of constraints "$D = v$" – v is picked randomly from all values of dimension D. More complex predicate types are tested in Section 7.5.

There are 20 such workloads: each one W.X.Y ($X = 1, 2, 3, 4$ and $Y = 0, 1, 2, 3, 4$) is the set of queries with X group-by dimensions and Y predicate dimensions generated in the above way. We randomly generate 50 queries for each workload W.X.Y. Note that for TPC-H, our workloads are different from the standard TPC-H queries, in order to mimic real-world analytical workloads and to cover more column combinations for each workload. Each standard TPC-H benchmark query (if it is an aggregation query) only covers one particular combination of dimensions and values.[2]

Performance metrics. We report both *Response Time* (ms) and *Speedup* of aggregation query processing in different system. Response time is the time to processing a query. Speedup is the relative ratio of response time in two systems.

Error metrics. We use the L^2 distance between the normalized estimated answer and the true one, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2$, to measure the *Error* in answers produced by different systems. For the ease of understanding, please refer to (2) in Section 2.2 for a more concrete form.

Parameter ϵ in our system specifies *Requested Error*, but the *Actual Error* of answers produced by AQP systems could be different.

Summary of results. We report the cost of constructing and storing samples and indexes in Section 7.2. We compare our SPS with DBX in Section 7.3. The actual error in answers produced by SPS is almost always lower than the requested error bound (specified by ϵ). When ϵ is 5%, SPS achieve a speedup of 100x for 90% of the queries in our workloads. For very selective queries, the speedup is less significant but still around 10x. When compared with other AQP systems SMG and BLK in Section 7.4, our SPS solution produces much less error when the response time is similar; and to produce answers with the same quality, SPS is much faster. The errors in the estimated answers given by SPS have much smaller standard deviations, which implies that the error is more likely to be bounded consistently in SPS. When applied on general aggregates with complex predicates in Section 7.5, our SPS still has the best performance gain compared to SMG and BLK.

[2]Although we use TPC-H in our experiments, the queries used in our experiments are quite different from standard TPC-H benchmarks. So our results do not reflect TPC-H benchmark numbers.

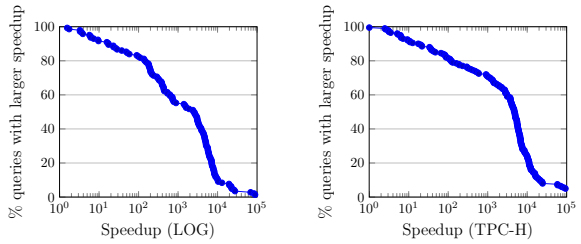


Figure 5: Speedup for the whole workload (twenty together)

7.2 Sampling Budget and Index Size

We first report the overhead of our samples/indexes in terms of both their sizes and construction time in Table 1.

Sampling budget and in-memory samples. Recall that the size of each sample is function of both the required error bound ϵ and the number of rows n, *i.e.*, $\sqrt{n}/\epsilon^2$. The *sampling rate* or *sampling budget* is $(\sqrt{n}/\epsilon^2)/n = (1/\sqrt{n}) \cdot (1/\epsilon^2)$. So one property of our system is that *the sampling rate/budget decreases when the data size increases*, which is a significant advantage for "big data."

The right part of Table 1 gives the sampling budgets (column "budget") and the total sizes of uniform and measure-biased samples (column "size") for different values of error bound ϵ in datasets of different sizes. Dictionary compression is applied on these samples. Because of the above property about sampling budget, the sample sizes increase very slowly for larger datasets for fixed ϵ.

On-disk indexes. Table 1 also gives the size of our measure-augmented indexes, denoted as "MA index" there. Its size is independent on the error bound ϵ, but roughly linear to the data size. The size of the low-frequency group indexes ("LF index" in Table 1) is also independent on ϵ, but is stable for larger datasets. This is because the selectivity threshold $s_0 = 1/\sqrt{n}$ decreases in larger datasets, so fewer values have "low-frequency."

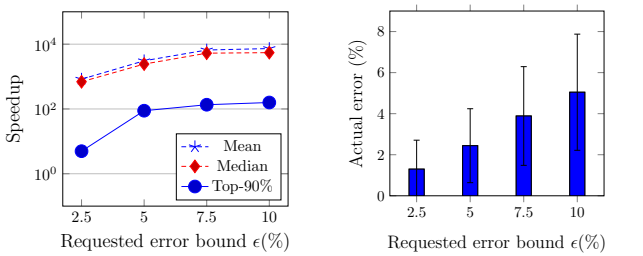
The total proportion of samples and indexes to the original data size decreases for larger datasets (from 1.08 in LOG-S to 0.41 in LOG for $\epsilon = 0.05$) – so our solution is more suitable for "big data."

Construction time. The time needed to build samples and indexes is independent on the error bound but linear to the data size (# rows and # columns), as samples and indexes are constructed during scans of tables. The building time is reported in Table 1 for each portion. The total building time (pre-processing) ranges from 0.7 hour for the smallest LOG-S to 2.4 hours for the largest LOG. Considering that one query in LOG may take several minutes to be completed in DBX and the performance gain of SPS, the cost of this one-time building process is reasonable and profitable.

7.3 Performance Gain over DBMS

We compare our SPS solution with a commercial database product DBX. For our workloads of aggregation queries, column-store indexes give the best performance in DBX, and the memory in our machine is large enough to allow DBX to cache column-store indexes in memory. For TPC-H, we run DBX on a view table that materializes all the foreign-key joins for DBX's best performance.

We compare SPS with DBX on all the twenty workloads W.X.Y (X $= 1 \ldots 4$ and Y $= 0 \ldots 4$) for both COUNT and SUM.



(a) Speedup over DBX (b) Actual error

Figure 6: Varying requested error bound ϵ (LOG)

Exp-I: Overall speedup. Figure 5 reports the overall speedup gained in our SPS system over DBX on all the twenty workloads for SUM aggregates in LOG and TPC-H. The x-axis in Figure 5 is the speedup ratio obtained by each query in our system, and the y-axis is the percentage of queries that obtain higher speedup. For example, a point $(10^2, 90)$ means 90% of all queries are sped up more than 100 times. We can find that, in both LOG and TPC-H, 90% queries in the workload get around 100x speedup or more, and the median speedup is over 10^3x and close to 10^4x.

The requested error bound ϵ is set to 0.05 here. As shown in Exp-II, the actual error is lower than ϵ most of the time.

Exp-II: Vary requested error bound ϵ. Such tremendous speedup is, of course, at the cost of error in the approximate answers produced by SPS. However, the error can be easily controlled by users using the parameter ϵ. We vary the request error bound ϵ, and report the speedups in Figure 6(a) and actual errors in Figure 6(b), where we use all the twenty LOG workloads with SUM aggregates. The error bounds ϵ requested by users are varied from 0.025 to 0.1.

Figure 6(a) plots 90-percentile / median / mean speedups. With larger ϵ, *i.e.*, if users can tolerate a bit larger error, they can gain more significant speedups. But even when they request the error to be no more than $\epsilon = 0.025$, the median / mean speedup is from 10^2 to 10^3. Figures 6(b) plots the average errors together with error bars representing one standard deviation, which are roughly the ranges of min/max error, for each ϵ. The average actual error (even plus standard deviation) is much lower than the requested error bound.

The speedups and errors for TPC-H have very similar trend to LOG. And SPS gets even lower actual errors for COUNT workloads. So those figures are omitted due to the space constraint.

Additional experiments. Exp-III (vary database size), Exp-IV (vary selectivity of queries), and Exp-V (vary number of dimensions in queries) are reported in Appendix D.

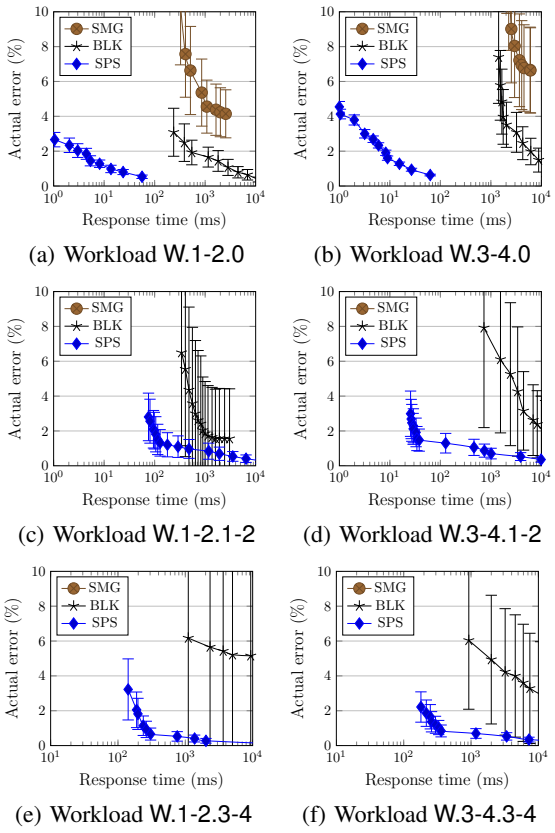
7.4 Comparing with Stratified Sampling

We compare our AQP system SPS with BLK and SMG. We use the real dataset LOG-S for this comparison.

Parameters and settings. For both BLK and SMG, we try different values of their parameters and choose the best ones in our experiments. In SMG, there is a threshold for the size of "small groups", and the best value we find in the testing dataset is 2000. In BLK, the similar parameter K (refer to Section 8 for a brief introduction) is set to be 10000 for the best performance. The space budget for samples in BLK is set to be 50% of the dataset size (the same as in [5]) so that we are able to load all the samples of BLK into memory for the fastest possible processing with these samples.

We give the distribution of the 20 SUM-workloads W.X.Y (X = 1-4 and Y = 0-4) (refer to the definitions Section 7.1) to BLK's optimization module which chooses the best set of stratified samples to cover them. Then we use the same workloads for comparison.

Comparison results and analysis. BLK can guarantee response time. For each query, we continuously enlarge the requested response time, so that the actual error in its answer can be reduced. SPS makes such tradeoff between time and error by varying ϵ and early termination. Similar tradeoff can be enabled in SMG by randomly permuting rows in the samples of SMG (in the preprocessing), so errors in SMG keep reducing as a query is being evaluated.



(a) Workload W.1-2.0 (b) Workload W.3-4.0

(c) Workload W.1-2.1-2 (d) Workload W.3-4.1-2

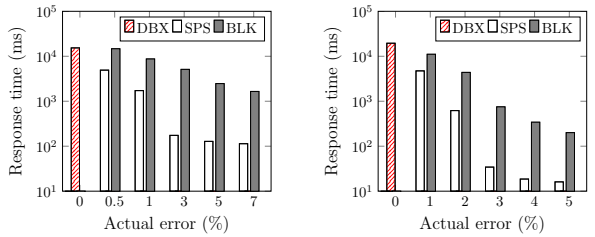
(e) Workload W.1-2.3-4 (f) Workload W.3-4.3-4

Figure 7: Time-error tradeoff in different systems

The 20 workloads are partitioned into six sets as in Figure 7. W.X_1-X_2.Y_1-Y_2 means all queries in W.X.Y with X = X_1 or X_2, and Y = Y_1 or Y_2. The comparison results for the three are reported in Figures 7(a)-7(f) for in-depth analysis. In each figure, one point (x, y) in a chart corresponds to "(the average response time, the average actual error)" of queries. The actual error is associated with an error bar with a width of one standard deviation to show how stable the actual error is for certain requested response time.

• **Queries with no predicates.** W.1-2.0 and W.3-4.0 (in Figures 7(a)-7(b)) are queries without predicates. All the three systems perform reasonably well for them, producing answers with error less than 0.1 within seconds. It is also obvious that our SPS has the narrowest error bar and the lowest average error for the same response time. This is because of our fundamentally different sampling scheme, measure-biased sampling, which produces precision guarantee independent on the underlying data distribution.

• **Queries with predicates.** For queries with predicates (the rest four sub-workloads in Figures 7(c)-7(f)), results about SMG are not plotted, because either its error is much higher than SPS and BLK, or its response time is too long. The reason is that SMG's small group sampling is a stratified sampling on only one dimension, so for selective predicates, its stratified samples can only capture a small number of rows satisfying the predicate, which makes its estimation inaccurate. We also observe that, SPS outperforms BLK a lot on error when the response time is similar. This is due to both our new sampling technique and the novel index structures. With the help of the indexes, our online sampling only needs $1/\epsilon^2$



(a) SUM(M) (b) COUNT(DISTINCT M)

Figure 8: With complex predicates and general aggregates

random seeks to produce an ϵ-approximation for any small query. On the other hand, when only very few rows satisfy a small query's predicate, if the predicate's column set is not covered exactly by the columns of any stratified sample selected by BLK's optimization module, BLK will suffer from the same problem as SMG (although it happens less frequently than in SMG).

7.5 Extensibility

We conduct experiments on more general workloads (generic aggregates with range constraints and complex logical expressions as predicates) to evaluate the extensibility of SPS as introduced in Section 6.2. SPS still outperforms the others on such workloads.

The workloads are generated on LOG-S with 1-4 group-by dimensions, and complex logical expressions on 4 dimensions as predicates, such as "$(C_1 = v_1 \text{ OR } v_2' \leq C_2 \leq v_2'') \text{ AND } (C_3 = v_3 \text{ OR } v_4' \leq C_4 \leq v_4'')$". We generate two such workloads with aggregates SUM(M) and COUNT(DISTINCT M), respectively. We always include 2 equality constraints on categorical dimensions and 2 range constraints on numerical dimensions in a predicate. For each of such workloads, dimensions, values, and logical expressions are randomly picked to form 200 queries.

Results for the two workloads are reported in Figure 8. We plot the average response time of SPS and BLK when achieving certain actual error, and compare them with DBX (we ignore SMG as it is dominated by the other two). Overall, our SPS provides much better tradeoff between time and error than BLK does, and is always faster than BLK when with the same error.

The performance gain of SPS over BLK and DBX on SUM is significant (Figure 8(a)) because of the strong theoretical guarantee. SPS can benefit from measure-biased samples on other aggregates AVG and STDEV as well even though the theoretical guarantees are weaker for them. It benefits from our samples and indexes also on COUNT(DISTINCT M) (Figure 8(b)). So when the error is above 2% or 1%, SPS is still the best option with significant performance gain and smooth tradeoff between time and accuracy. In particular, SPS still gets 100x speedup when the error is 5%.

8. RELATED WORK

AQP has been studied extensively during last few decades. There are three related lines, including i) online sampling – drawing samples after each query comes, ii) offline sampling – drawing samples before queries come; and iii) non-sampling based approaches.

Online sampling/aggregation. Online aggregation [19, 14, 24] streams data in a random order or assumes that sample rows can be randomly drawn from the underlying database, and uses them to estimated query answers. The disadvantage is that drawing rows randomly from the table for each query means many random I/O accesses, which are costly at query time, resulting in high latency. Such techniques are orthogonal to our ideas of sampling and building indexes before queries come, and can be integrated into our system to provide estimations before index building is finished.

Offline sampling. This line of work is especially relevant to ours. The idea is to create random samples before queries come, and calibrate their sampling rates carefully using stratified (or importance) sampling [22] to reduce variance when using them to answer queries. There are two types of offline sampling techniques: one calibrates sampling rates without the knowledge of workload; and the other utilizes historical workload distribution.

• **Workload-independent sampling.** Aqua [3, 2] proposes congressional sampling. For all possible queries, consider combinations of group-by dimensions. For each combination, the sampling rate is set based on a weighting function that considers the group sizes for all subsets of the group-by dimensions in this combination. For d possible group-by dimensions, there are 2^d combinations, which renders it impractical for datasets with dozens or hundreds of possible group-by columns. In addition, Aqua operates on a single sample which has to be very general-purpose in nature but only loosely appropriate for any particular query [7].

To overcome such shortcomings, Babcock *et al.* [7] proposes small group sampling, a stratified sampling technique that builds a biased sample on each single column. Uniform sampling does a satisfactory job at providing good estimates for the large groups in a group-by query. A group here is the set of all rows with certain (the same) value on a group-by dimension. For small groups, uniform sampling fails to draw enough number of rows for each. So all the rows from the small groups are included in the biased sample on their dimension. A query is executed on the union of the uniform sample and all biased samples whose dimensions appear in the query for an estimated answer. Using similar heuristics, outlier indexing [11] is proposed to include rows with outlier values in additional samples to improve estimation accuracy.

• **Workload-aware sampling.** Workload information can be used to optimize samples' constitution. For example, Aqua [2] can be adapted to workloads. Workloads are also used in [15] to construct "self-tuning" biased samples. STRAT [12, 13] aims to minimize the expected relative error of workloads. [23] requires building a new sample for each query template. SciBORQ [27] targets scientific analysis and creates samples based on past query results with no guarantees on error margin. However, the assumption of these works that workloads are stable can easily be wrong in practice.

BlinkDB [5] abstracts workload to query column set (QCS), *i.e.*, the set of columns that appear in a query, allowing it to tolerate moderate workload change. A group is the set of all rows with the same values on a QCS. A stratified sample can be created for each QCS by sampling K rows for each group on this QCS – if a group has less than K rows, all of its rows are put in the sample. BlinkDB formulates an optimization problem to decides how to allocate space budget across QCSs based on previous workloads. It provides either time guarantee or CI-based error guarantee.

Related work on requested error v.s. actual error, non-sampling approaches, and other AQP systems is discussed in Appendix E.

9. CONCLUSIONS

We propose an AQP system based on a novel sample+seek framework. Distribution precision is guaranteed in answers to aggregation queries. A new measure-biased sampling technique is introduced to approximately process SUM aggregation queries with less selective predicates and achieve this precision guarantee. For queries with more selective predicates, sampling is not enough, and we propose two novel indexes to aid in-memory samples. The number of random seeks using our indexes can be bounded to achieve the same precision guarantee. Thus, our system is efficient for queries with various selectivities. Our system can extend to support a generic range of aggregates with complex predicates.

10. REFERENCES

[1] 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.

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

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

[4] 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.

[5] 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.

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

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

[8] P. Bille, A. Pagh, and R. Pagh. Fast evaluation of union-intersection expressions. In *ISAAC*, 2007.

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

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

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

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

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

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

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

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

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

[18] 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.

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

[20] 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.

[21] A. Kleiner, A. Talwalkar, S. Agarwal, I. Stoica, and M. I. Jordan. A general bootstrap performance diagnostic. In *KDD*, 2013.

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

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

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

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

[26] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys*, 1984.

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

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

[29] 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.

APPENDIX

A. PROOFS

Proof of Theorem 3. Suppose there are r groups in the answer $\mathbf{x}$ to Q. For each group $i \in \{1, \ldots, r\}$, let n_i be the number of rows belonging to the ith group and satisfying the predicate $\mathcal{P}$, and $n_{\mathcal{P}}$ be the total number of rows satisfying $\mathcal{P}$. So of course, $\sum_{i=1}^{r} n_i = n_{\mathcal{P}}$ and $s(\mathcal{P}) = n_{\mathcal{P}}/n$.

We draw a uniform sample of m rows T^0 from T. Similarly, in the sample T^0, let m_i be the number of rows belonging to the ith group and satisfying the predicate $\mathcal{P}$, and $m_{\mathcal{P}}$ be the number of rows satisfying $\mathcal{P}$.

Consider the answer $\mathbf{x}$ to Q and the estimated answer $\hat{\mathbf{x}}$, we have $x_i = n_i/n_{\mathcal{P}}$ and $\hat{x}_i = m_i/m_{\mathcal{P}}$. So,

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \mathbf{E}\left[\sum_{i=1}^{r}\left(\frac{n_i}{n_{\mathcal{P}}} - \frac{m_i}{m_{\mathcal{P}}}\right)^2\right] \tag{8}$$

$$= \mathbf{E}\left[\frac{n^2}{m^2 n_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(m\cdot\frac{n_i}{n} - m\cdot\frac{n_{\mathcal{P}}}{n}\cdot\frac{m_i}{m_{\mathcal{P}}}\right)^2\right]. \tag{9}$$

$m_{\mathcal{P}}$ can be interpreted as the number of successes in m independent Bernoulli trials with success probability $n_{\mathcal{P}}/n$. So $\mathbf{E}[m_{\mathcal{P}}] = m\cdot n_{\mathcal{P}}/n \triangleq \bar{m}_{\mathcal{P}}$. Similarly, m_i can be interpreted as the number of successes in m independent Bernoulli trials with success probability n_i/n. So $\mathbf{E}[m_i] = m\cdot n_i/n \triangleq \bar{m}_i$. From Chernoff bound, for $0 < \epsilon < 1$, we have

$$\mathbf{Pr}\left[m_{\mathcal{P}} \geq (1+\epsilon)\bar{m}_{\mathcal{P}}\right] \leq e^{-\epsilon^2\bar{m}_{\mathcal{P}}/3}, \tag{10}$$

$$\text{and } \mathbf{Pr}\left[m_{\mathcal{P}} \leq (1-\epsilon)\bar{m}_{\mathcal{P}}\right] \leq e^{-\epsilon^2\bar{m}_{\mathcal{P}}/2}. \tag{11}$$

Consider two events: $\mathsf{E}_{\mathsf{in}} = [(1-\epsilon)\bar{m}_{\mathcal{P}} < m_{\mathcal{P}} < (1+\epsilon)\bar{m}_{\mathcal{P}}]$ and $\mathsf{E}_{\mathsf{out}} = [m_{\mathcal{P}} \geq (1+\epsilon)\bar{m}_{\mathcal{P}} \vee m_{\mathcal{P}} \leq (1-\epsilon)\bar{m}_{\mathcal{P}}]$. We can then rewrite (9) using conditional probability:

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \mathbf{E}\left[\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\bar{m}_i - \bar{m}_{\mathcal{P}}\cdot\frac{m_i}{m_{\mathcal{P}}}\right)^2\right] = \tag{12}$$

$$\mathbf{E}\left[\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\bar{m}_i - \bar{m}_{\mathcal{P}}\cdot\frac{m_i}{m_{\mathcal{P}}}\right)^2\middle|\mathsf{E}_{\mathsf{in}}\right]\cdot\mathbf{Pr}\left[\mathsf{E}_{\mathsf{in}}\right] \tag{13}$$

$$+\,\mathbf{E}\left[\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\bar{m}_i - \bar{m}_{\mathcal{P}}\cdot\frac{m_i}{m_{\mathcal{P}}}\right)^2\middle|\mathsf{E}_{\mathsf{out}}\right]\cdot\mathbf{Pr}\left[\mathsf{E}_{\mathsf{out}}\right]. \tag{14}$$

Let $m_{\mathcal{P}}/\bar{m}_{\mathcal{P}} = \alpha$. (13) can be upper bounded as:

$$(13) \leq \mathbf{E}\left[\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\bar{m}_i - \frac{m_i}{\alpha}\right)^2\middle|1-\epsilon<\alpha<1+\epsilon\right] \tag{15}$$

$$\leq \max_{1-\epsilon<\alpha<1+\epsilon}\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\mathbf{E}\left[\left(\bar{m}_i - \frac{m_i}{\alpha}\right)^2\right] \tag{16}$$

$$= \max_{1-\epsilon<\alpha<1+\epsilon}\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\frac{1}{\alpha^2}\mathbf{Var}\left[m_i\right] + \frac{(\alpha-1)^2}{\alpha^2}\bar{m}_i^2\right) \tag{17}$$

$$\leq \frac{2}{\bar{m}_{\mathcal{P}}^2}\left(\sum_{i=1}^{r}\mathbf{Var}\left[m_i\right] + \sum_{i=1}^{r}\epsilon^2\bar{m}_i^2\right)\ (\epsilon \leq 0.5 \text{ is small}). \tag{18}$$

Since m_i is the number of successes in m Bernoulli trials,

$$\sum_{i=1}^{r}\mathbf{Var}\left[m_i\right] = \sum_{i=1}^{r} m\cdot\frac{n_i}{n}\cdot\left(1-\frac{n_i}{n}\right) \leq \frac{m\cdot n_{\mathcal{P}}}{n} = \bar{m}_{\mathcal{P}}. \tag{19}$$

From Cauchy-Schwarz inequality, we have

$$\sum_{i=1}^{r} \epsilon^2 \bar{m}_i^2 \le \epsilon^2 \left(\sum_{i=1}^{r} \bar{m}_i\right)^2 = \epsilon^2 \cdot \bar{m}_{\mathcal{P}}^2 \tag{20}$$

Putting (19) and (20) back to (18), we have

$$(13) \le \frac{2}{\bar{m}_{\mathcal{P}}} + 2\epsilon^2. \tag{21}$$

Now we upper bound (14). From (10) and (11),

$$\mathbf{Pr}\left[\mathsf{E}_{\mathsf{out}}\right] \le e^{-\epsilon^2 \bar{m}_{\mathcal{P}}/3} + e^{-\epsilon^2 \bar{m}_{\mathcal{P}}/2} \le 2e^{-\epsilon^2 \bar{m}_{\mathcal{P}}/3}. \tag{22}$$

For the values of $m_{\mathcal{P}}/\bar{m}_{\mathcal{P}} = \alpha$ in event $\mathsf{E}_{\mathsf{out}}$, we have

$$\mathbf{E}\left[\frac{1}{\bar{m}_{\mathcal{P}}^2}\sum_{i=1}^{r}\left(\bar{m}_i - \bar{m}_{\mathcal{P}} \cdot \frac{m_i}{m_{\mathcal{P}}}\right)^2 \middle| \mathsf{E}_{\mathsf{out}}\right] \tag{23}$$

$$\le \frac{1}{\bar{m}_{\mathcal{P}}^2}\mathbf{E}\left[\sum_{i=1}^{r}\bar{m}_i^2 + \sum_{i=1}^{r}\left(\bar{m}_{\mathcal{P}} \cdot \frac{m_i}{m_{\mathcal{P}}}\right)^2\right] \le 2. \tag{24}$$

where the second inequality of (24) is from Cauchy-Schwarz inequality and linearity of expectation. Putting (22) and (24) back to (14), together with (13) and (21), we have

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] \le \frac{2}{\bar{m}_{\mathcal{P}}} + 2\epsilon^2 + 4e^{-\epsilon^2 \bar{m}_{\mathcal{P}}/3} \le 6\epsilon^2, \tag{25}$$

if $\bar{m}_{\mathcal{P}} = \Omega\big(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon^2}\big)$, and from Jensen's inequality,

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2\right] \le \sqrt{\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right]} \le 3\epsilon. \tag{26}$$

We can conclude the proof using a generalized McDiarmid's inequality. Rewrite $\phi(t_1, \ldots, t_m) \triangleq \|\mathbf{x} - \hat{\mathbf{x}}\|_2$, where t_i is a sample row from the table T and $\{t_i \mid i = 1, \ldots, m\}$ are mutually independent. Consider a different estimation $\hat{\mathbf{x}}' = \langle \hat{x}'_1, \ldots, \hat{x}'_r \rangle$, where $\hat{x}'_i = m_i/\bar{m}_{\mathcal{P}}$ (and truncate $\hat{x}'_i$ at 1). Note that $\bar{m}_{\mathcal{P}}$ is unknown so $\hat{\mathbf{x}}'$ is only conceptually feasible. Similarly, let $\phi'(t_1, \ldots, t_m) \triangleq \|\mathbf{x} - \hat{\mathbf{x}}'\|_2$. Using Chernoff bound as (10), if $\bar{m}_{\mathcal{P}} = \Omega\big(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\big)$,

$$\mathbf{Pr}\left[1 - \epsilon \le \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x} - \hat{\mathbf{x}}'\|_2} \le 1 + \epsilon\right] \ge 1 - \delta. \tag{27}$$

For any two random samples $\{t_1, \ldots, t_i, \ldots, t_m\}$ and $\{t_1, \ldots, t'_i, \ldots, t_m\}$ differing at only one pair of tuples t_i and t'_i, the function ϕ' satisfies the Lipschitz property that

$$|\phi'(t_1, \ldots, t_i, \ldots, t_m) - \phi'(t_1, \ldots, t'_i, \ldots, t_m)| = c_i \le 2/\bar{m}_{\mathcal{P}}.$$

and $\sum_{i=1}^{m} c_i^2 \le 4/\bar{m}_{\mathcal{P}}$. So using McDiarmid's inequality,

$$\mathbf{Pr}\left[\|\mathbf{x} - \hat{\mathbf{x}}'\|_2 > \mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}'\|_2\right] + \epsilon\right] \le e^{-8\epsilon^2 \bar{m}_{\mathcal{P}}} = \delta. \tag{28}$$

Thus, from (26), (27), and (28), we have

$$\mathbf{Pr}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \le \epsilon\right] \ge 1 - \delta, \tag{29}$$

if $m = \Omega\Big(\frac{1}{s_0} \cdot \frac{1}{\epsilon^2} \cdot \big(\log\frac{1}{\epsilon^2} + \log\frac{1}{\delta}\big)\Big)$ (note $\bar{m}_{\mathcal{P}} = m \cdot s_0$). □

Proof of Theorem 5. Let's first focus on a special case where $s_0 = 1$ and the query $Q(\mathcal{G}, \mathsf{SUM}(M), \varnothing)$ has no predicate.

Suppose there are r groups in the answer $\mathbf{x} = \langle x_1, \ldots, x_r \rangle$ to Q. Let $T_i \subseteq T$ be the set of rows belonging to the ith group for $i = 1, \ldots, r$. Recall that, for a set T of rows, $M(T) = \sum_{t \in T} t_M$ is the sum of values of measure M over all rows in T. So we have $x_i = M(T_i)/M(T)$.

Let m be the number of rows in the measure-biased sample T^M, and let m_i be the number of rows belonging to the ith group in T^M. Recall that, in our estimated answer $\hat{\mathbf{x}} = \langle \hat{x}_1, \ldots, \hat{x}_r \rangle$, we have $\hat{x}_i = m_i/m$. So we can rewrite

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \frac{1}{m^2}\sum_{i=1}^{r}\mathbf{E}\left[\left(\frac{M(T_i)}{M(T)} \cdot m - m_i\right)^2\right]. \tag{30}$$

In our measure-biased sampling, a row in the ith group T_i is drawn into T^M with probability $p_i^M = M(T_i)/M(T)$. So m_i is the number of successes of in m independent Bernoulli trials with success probability p_i^M. We have $\mathbf{E}[m_i] = mp_i^M$ and $\mathbf{Var}[m_i] = mp_i^M(1 - p_i^M)$. (30) can be rewritten as

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \frac{1}{m^2}\sum_{i=1}^{r}\mathbf{Var}[m_i] = \frac{1}{m}\sum_{i=1}^{r}p_i^M(1 - p_i^M) \le \frac{1}{m}.$$

The rest part is similar to the proof of Theorem 3. Let $m = \Omega\big((1/\epsilon^2) \cdot \log(1/\delta)\big)$. Using Jensen's inequality, we could have $\mathbf{E}[\|\mathbf{x} - \hat{\mathbf{x}}\|_2] \le 1/\sqrt{m} \le \epsilon$. As replacing one row in T^M changes $\|\mathbf{x} - \hat{\mathbf{x}}\|_2$ at most by $1/m$, using McDiarmid's inequality, $\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \le 2\epsilon$ with probability at most $1 - \delta$.

Now consider the general case with $0 < s_0 \le 1$ and a query $Q(\mathcal{G}, \mathsf{SUM}(M), \mathcal{P})$ with measure selectivity $s_M(\mathcal{P}) \ge s_0$. The idea is identical to the proof of Theorem 3 so we only sketch it in the following part.

Let $T_{\mathcal{P}} \subseteq T$ be the set of rows satisfying $\mathcal{P}$ in T, and $T_i \subseteq T_{\mathcal{P}}$ be the set of rows satisfying $\mathcal{P}$ and belonging to the ith group in T. Also, let $m_{\mathcal{P}}$ be the number of rows satisfying $\mathcal{P}$ in the sample T^M, and m_i be the number of rows satisfying $\mathcal{P}$ and belonging to the $i$the group in T^M.

In our measure-biased sampling, it is not hard to see $\mathbf{E}[m_{\mathcal{P}}] = m \cdot M(T_{\mathcal{P}})/M(T) \triangleq \bar{m}_{\mathcal{P}}$. Similar to the proof of Theorem 3, consider two events: $\mathsf{E}_{\mathsf{in}} = [(1 - \epsilon)\bar{m}_{\mathcal{P}} < m_{\mathcal{P}} < (1 + \epsilon)\bar{m}_{\mathcal{P}}]$ and $\mathsf{E}_{\mathsf{out}} = [m_{\mathcal{P}} \ge (1 + \epsilon)\bar{m}_{\mathcal{P}} \vee m_{\mathcal{P}} \le (1 - \epsilon)\bar{m}_{\mathcal{P}}]$. Decompose $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right]$ as in (13)-(14) into

$$\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 \mid \mathsf{E}_{\mathsf{in}}\right] \cdot \mathbf{Pr}\left[\mathsf{E}_{\mathsf{in}}\right] \tag{31}$$

$$+ \mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 \mid \mathsf{E}_{\mathsf{out}}\right] \cdot \mathbf{Pr}\left[\mathsf{E}_{\mathsf{out}}\right]. \tag{32}$$

We can show that $\mathbf{Pr}\left[\mathsf{E}_{\mathsf{out}}\right]$ is small enough so that (32) only adds a term ϵ. On the other hand, in the event E_{in}, $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 \mid \mathsf{E}_{\mathsf{in}}\right]$ can be upper bounded similarly as in the above special case by $1/\bar{m}_{\mathcal{P}}$ with high probability. Putting them together with a generalized argument using McDiarmid's inequality completes the proof. □

Proof of Proposition 7. In the answer $\mathbf{x} = \langle x_1, \ldots, x_r \rangle$ to Q, there are r groups. Let $T_i \subseteq T_{\mathcal{P}}$ be the set of rows belonging to the ith group for $i = 1, \ldots, r$. For a set T of rows, recall $M(T) = \sum_{t \in T} t_M$. So we have $x_i = M(T_i)/M(T_{\mathcal{P}})$. Let m be the number of rows in $T'_{\mathcal{P}}$ and T'_i be the set of rows belonging to the ith group in $T'_{\mathcal{P}}$. We estmate $\hat{x}_i = M(T'_i)/M(T'_{\mathcal{P}})$.

We rewrite $\mathbf{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] =$

$$= \frac{1}{M(T'_{\mathcal{P}})^2}\sum_{i=1}^{r}\mathbf{E}\left[\left(x_i \cdot M(T'_{\mathcal{P}}) - M(T'_i)\right)^2\right] \tag{33}$$

$$\le \frac{1}{m^2}\sum_{i=1}^{r}\mathbf{E}\left[\left(x_i \cdot M(T'_{\mathcal{P}}) - M(T'_i)\right)^2\right]. \tag{34}$$

Now focus on the ith group. Consider one random draw of a row t in $T'_{\mathcal{P}}$. Let $\mathrm{I}_{t,i}$ be indicator variable of the event that $t \in T_i$ (*i.e.*, $\mathrm{I}_{t,i} = 1$ if $t \in T_i$ and $\mathrm{I}_{t,i} = 0$ otherwise). Define random variable $Z_{t,i} = x_i \cdot t_M - t_M \cdot \mathrm{I}_{t,i}$. We can rewrite the term in (34) as $x_i \cdot M(T'_{\mathcal{P}}) - M(T'_i) = \sum_{t \in T'_{\mathcal{P}}} Z_{t,i}$. It is not hard to see that $\mathbf{E}_t[Z_{t,i}] = 0$. And because the m rows in $T'_{\mathcal{P}}$ are drawn

independently, we have

$$\mathbf{E}\left[\left(\Sigma_{t\in T'_{\mathcal{P}}} Z_{t,i}\right)^2\right] = m\mathbf{E}_t\left[Z_{t,i}^2\right].$$

Putting it back to (34), we have

$$\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] \le \frac{1}{m}\sum_{i=1}^{r}\mathbf{E}_t\left[Z_{t,i}^2\right]. \tag{35}$$

We upper bound $\mathbf{E}_t\left[Z_{t,i}^2\right]$ in the rest part.

$$\mathbf{E}_t\left[Z_{t,i}^2\right] = \mathbf{E}_t\left[x_i^2\cdot t_M^2 - 2x_i\cdot t_M^2\cdot \mathrm{I}_{t,i} + t_M^2\cdot \mathrm{I}_{t,i}^2\right]. \tag{36}$$

Let $n = |T_{\mathcal{P}}|$ and $n_i = |T_i|$. Using the fact that $t_M \in [1,\Delta]$ for any row t, we have

$$\mathbf{E}_t\left[x_i^2\cdot t_M^2\right] = x_i^2\cdot\sum_{t\in T_{\mathcal{P}}}\frac{t_M^2}{n} \le x_i^2\cdot\frac{n\Delta^2}{n} = x_i^2\Delta^2.$$

And similarly, from $n_i/n \le x_i\Delta$, we have

$$\mathbf{E}_t\left[t_M^2\cdot \mathrm{I}_{t,i}^2\right] = \sum_{t\in T_i}\frac{t_M^2}{n} \le \frac{n_i\Delta^2}{n} \le x_i\Delta^3,$$

Putting them back to (36), we have

$$\mathbf{E}_t\left[Z_{t,i}^2\right] \le x_i^2\Delta^2 + x_i\Delta^3 \le (\Delta^3+\Delta^2)\cdot x_i. \tag{37}$$

And putting (37) back to (35), we have

$$\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] \le \frac{1}{m}\sum_{i=1}^{r}(\Delta^3+\Delta^2)\cdot x_i = \frac{\Delta^3+\Delta^2}{m}.$$

Let $m = \Omega\left(\Delta^3/\epsilon^2\right)$ and then we have $\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] \le \epsilon^2$, and from Jessen's inequality, we complete the proof. □

Proof of Theorem 8. In the answer $\mathbf{x} = \langle x_1,\ldots,x_r\rangle$ to Q, there are r groups. Let $T_i \subseteq T_{\mathcal{P}}$ be the set of rows belonging to the ith group. Let $n = |T_{\mathcal{P}}|$ and $n_i = |T_i|$. For a set T of rows, recall $M(T) = \sum_{t\in T} t_M$. So we have $x_i = M(T_i)/M(T_{\mathcal{P}})$.

Let m be the number of rows in the approximate measure-biased sample $T'_{\mathcal{P}}$, and let T'_i be the set of rows belonging to the ith group in $T'_{\mathcal{P}}$. Recall that we estimate x_i as $\hat{x}_i = M'(T'_i)/M'(T'_{\mathcal{P}})$. We can rewrite $\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] =$

$$= \frac{1}{M'(T'_{\mathcal{P}})^2}\sum_{i=1}^{r}\mathbf{E}\left[\left(x_i\cdot M'(T'_{\mathcal{P}}) - M'(T'_i)\right)^2\right]. \tag{38}$$

(38) is analogous to (33). The probability of drawing a row from T_i is $p_i = \sum_{t\in T_i}\mathsf{apx}(t_M)/\sum_{t\in T_{\mathcal{P}}}\mathsf{apx}(t_M)$. It is not hard to show $p_i \le 2x_i$. Similar to (33)-(37), we can show

$$\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] \le \frac{12}{m},$$

with $\Delta = 2$. So with $m = \Omega\left(1/\epsilon^2\right)$, we have $\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2\right] \le \epsilon^2$, and from Jessen's inequality, we have $\mathbf{E}\left[\|\mathbf{x}-\hat{\mathbf{x}}\|_2\right] \le \epsilon$.

Finally, replace one row in $T_{\mathcal{P}}$ changes $\|\mathbf{x}-\hat{\mathbf{x}}\|_2$ at most by $2/m$ (because $1 \le t_{M'} < 2$). So, using McDiarmid's inequality, we have that $\|\mathbf{x}-\hat{\mathbf{x}}\|_2 \le 2\epsilon$, with probability at most $1-\delta$, if $m = \Omega\left((1/\epsilon^2)\cdot\log(1/\delta)\right)$. □

B. DRAWING SAMPLES EFFICIENTLY

In the first scan, for each measure M, we compute $M(T) = \sum_{t\in T} t_M$, *i.e.*, the sum of values on M over all rows in T. We generate m random real numbers $S^M(m)$ uniformly from the range $[0, M(T))$. Then in the second scan, we maintain a partial sum of measure values when scanning rows; more specifically, for each row t, we maintain the sum of t'_M for all rows t' that were scanned before arriving at t, denoted as $\sigma_M(t)$. We add one copy of t into T^M if there exists $s \in S^M(m)$ s.t. $\sigma_M(t) \le s < \sigma_M(t) + t_M$, and k copies of t if there are k such samples s in $S^M(m)$.

The intuition behind the above implementation is that a row t occupies an interval $[\sigma_M(t), \sigma_M(t)+t_M)$ with width t_M on the whole range $[0, M(T))$, so a uniformly picked number from the range falls into the interval of t with probability exactly proportional to t_M (as required in (4) by measure-biased sampling). A simplified version can be used for generating uniform samples.

We illustrate this sampling procedure using an example.

EXAMPLE B.1. (Continue Example 1.1) *Let's use the table T with 200 rows in Figure 1(a) and measure M to demonstrate how to implement measure-biased sampling in two scans. To draw a measure-biased sample T^M with 20 rows, in the first scan, we calculate $M(T) = \sum_t t_M = 488$. We draw 20 random numbers $S^M(20)$ uniformly from the range $[0, 488]$. For example, we could have $S^M(20) = \{0, 25, 50, \ldots, 450, 475\}$.*

We maintain partial sums on M during the second scan of rows from 1 to 200. For row 26, the partial sum is 25 and we have 25 in $S^M(20)$, so row 26 is put into T^M. For row 199, the partial sum is 288 and $\{300, 325, 350, 375\} \subseteq S^M(20)$ falling into the range $[288, 388)$, so 4 copies of row 199 are put into T^M. Figure 9 shows how the measure-biased sample T^M is constructed from random numbers in $S^M(20)$. When there are multiple measures, the second scans for all of them can be easily merged (by maintaining partial sums on all the measures simultaneously).

C. LOW-FREQUENCY GROUP INDEX

Construction. For a dimension D and a value v, let $T_{D=v}$ be the set of rows with value v on dimension D. In a table of n rows, it is obvious that if there is an equi-constraint $D_i = v_i$ in the predicate of Q with $|T_{D_i=v_i}| \le \sqrt{n}$ (or $n\cdot s_0$), then Q must be a (s_0-)small query. For each (dimension,value)-pair (D, v), iff $|T_{D=v}| \le \sqrt{n}$, the low-frequency group index *materializes* $T_{D=v}$, *i.e.*, explicitly stores rows in $T_{D=v}$ sequentially. Note that in our measure-augmented inverted index, $\mathsf{inv}(D, v)$ points to the same set of rows as $T_{D=v}$ does. But $\mathsf{inv}(D, v)$ keeps only row IDs and approximate measures, while in our low-frequency group index, $T_{D=v}$, if presents, keeps these rows with all dimensions and measure attributes in a row-oriented format.

We cannot get a tight theoretical bound of the size of this index as the set of pairs (D, v) with $|T_{D=v}| \le \sqrt{n}$ is dependent on the frequency distribution of dimension D. But since $\sqrt{n}$ is small, in both benchmark and real datasets, we observe that its size is usually very small (see "LF index" in Table 1). We store it on disk. When the disk space budget is a problem, we can choose to not build it, as measure-augmented inverted index suffices for all small queries.

Processing queries. For a query Q, we check two conditions: whether i) its predicate $\mathcal{P}$ is in the form of (7) and ii) $\mathcal{P}$ contains an equi-constraint "$D_i = v_i$" with $T_{D_i=v_i}$ stored in the index. If yes, we simply need to scan up to $\sqrt{n}$ rows in $T_{D_i=v_i}$ sequentially to get the exact answer. It corresponds to $\mathsf{ProcessWithLFIndex}(Q)$ in lines 8-9 of Algorithm 1. For example, the following query

SELECT C_1, SUM(M) FROM T GROUP BY C_1
WHERE $C_2 = 1$ AND ($C_3 = 0$ OR $C_3 = 1$)

can be answered by scanning rows $T_{C_2=1}$.

THEOREM 10. *If a query Q satisfies the above conditions i) and ii), the low-frequency group index can be used to answer Q. On*

t_{ID}	1	26	51	76	92	94	96	98	111	136	161	186	199	200
Partial sum $\sigma_M(t)$	0	25	50	75	100	120	150	170	200	225	250	275	288	388
Numbers in $S^M(20)$	0	25	50	75	100	125	150	175	200	225	250	275	300, 325, 350, 375	400, 425, 450, 475
$\sigma_M(t) + t_M$	1	26	51	76	110	130	160	180	201	226	251	276	388	488
# copies of t in T^M	1	1	1	1	1	1	1	1	1	1	1	1	4	4

Figure 9: Drawing a Measure-biased Sample T^M

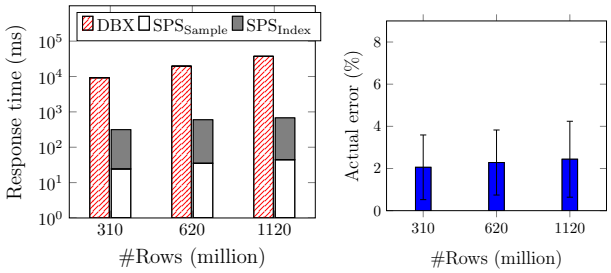
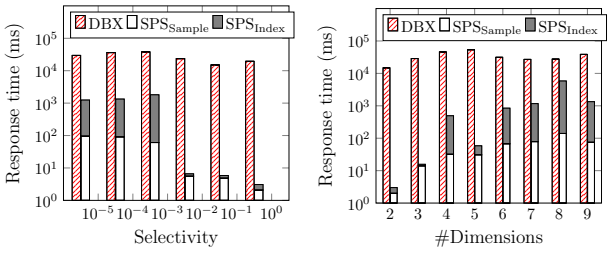


Figure 10: Varying number of rows (LOG)



(a) Varying selectivity (b) Varying # dimensions

Figure 11: Selectivity and # dimensions of queries (LOG)

a table of n rows, we need to retrieve at most $\sqrt{n}$ rows sequentially from the disk to compute its answer.

D. ADDITIONAL EXPERIMENTS

Exp-III: Vary database size. We test the scalability of our system and compare it with DBX for datasets with different sizes. We use datasets LOG-S, LOG-M, and LOG (refer to Table 1) and set $\epsilon = 0.05$. Average response time (in DBX and SPS) and average actual error with one standard deviation (in SPS) for queries in all the twenty workloads are plotted in Figure 10. First of all, we find that the actual error is quite stable for different databases, which is not surprising because of our rigorous theoretical guarantees.

For response time of SPS, we decompose it into $\mathsf{SPS}_{\mathsf{Sample}}$ and $\mathsf{SPS}_{\mathsf{Index}}$, which represent the time SPS spends on in-memory samples and on-disk indexes, respectively. It can be seen that response time in SPS increases slightly for larger datasets, but the $\mathsf{SPS}_{\mathsf{Index}}$ part is quite stable. That is because the number of random accesses needed in our algorithm is independent on the database size (Theorem 8). On the other hand, DBX's response time increases linearly to the number of rows, because column-store indexes are used.

Exp-IV: Vary selectivity of queries. Selectivity (ratio of # rows satisfying the predicate to total # rows) may affect the response time of our system SPS. It is because large queries (with selectivity higher than $1/\sqrt{n}$) only need to scan in-memory samples, but small queries may also need to use on-disk indexes. So we partition queries in all the twenty workloads into six buckets with selectivities in $(10^{-1}, 1]$, $(10^{-2}, 10^{-1}]$, $\ldots$, $(0, 10^{-5}]$. Average response time in each bucket is plotted in Figures 11(a) for LOG with $\epsilon = 0.05$. The one for TPC-H looks quite similar.

We find that $\mathsf{SPS}_{\mathsf{Sample}}$ increases as selectivity becomes smaller. It is because of the early termination condition when scanning samples: an ϵ-approximation $\hat{\mathbf{x}}$ needs to collect $1/\epsilon^2$ rows that satisfies the query predicate from the sample – if the predicate has large selectivity, this condition can be satisfied earlier in the scan. $\mathsf{SPS}_{\mathsf{Index}}$ tends to be stable for small selectivity as the number of random accesses is upper bounded by both $1/\epsilon^2$ (Theorem 9).

Average actual error is steadily below the requested bound 0.05 because of our rigorous theoretical guarantee.

Exp-V: Vary number of dimensions in queries. We partition queries into buckets based on the numbers of dimensions (in group-bys, predicates, and measure of each query). In our workloads, the number of dimensions varies from 2 to 9. Average response time in each bucket is plotted in Figure 11(b) for LOG with $\epsilon = 0.05$. The one for TPC-H is similar and thus omitted here. With more dimensions, the response time of SPS increases because of two reasons: i) smaller selectivity (so $\mathsf{SPS}_{\mathsf{Index}}$ increases); and ii) more columns to be scanned (so $\mathsf{SPS}_{\mathsf{Sample}}$ increases). SPS always has significant speedup over DBX and tends to be stable after six dimensions.

E. ADDITIONAL RELATED WORK

Requested error v.s. actual error. Most of sampling-based systems estimate error bars using central limit theorem (CLT), Hoeffding inequality [6], or bootstrap [29]. In particular, BlinkDB [5] estimates error bar using standard closed-form formulas during online query. It is then claimed in [4] that the estimated error bar fails a lot in real query workload (also refer to our Example 1.1). A recent diagnostic algorithm (Kleiner *et al.* [21]) is extended by [4] to diagnose the failure of error bar estimation.

So requested (or estimated) error is not always below the actual error. That is also one motivation why we introduce a more rigorous notion, distribution predication guarantee, in this work. As far as we are concerned, our work is the first to provide such theoretical guarantees in AQP. One of our theoretical result on uniform sampling is inspired by the result in Acharya *et al.* [1]. It proves that uniform sampling achieves a tight bound in histogram approximation, which can be applied in our problem when the query has no predicate and the aggregate function is COUNT.

Non-sampling based approaches. Another orthogonal line of work are view materialization [6] and datacube [18] which precompute answers to some queries and create summaries to accelerate OLAP query processing. Histograms [17] and wavelets [28, 9, 17] create compressed synopses of relational tables to support some aggregates. These techniques either have too large space overhead (super-linear) to be applicable for enterprise-scale workloads and datasets, or are applicable for only a subclass of aggregates supported by ours (*e.g.*, without joins and complex predicates).

Other AQP systems. Deterministic approximate querying (DAQ) schemes are recently proposed in [25]. It proposes novel bit-sliced indexes to store columns separately so that a query can be evaluated on the first few bits while ignoring the remaining bits for each row. However, the bitwise DAQ still needs to exhaust all bits when doing equality check on categorical dimensions, and in general it needs to scan all rows for the most significant bits, so the speedup is limited for the query class we support. However, it would be a very interesting line of future work to incorporate such bit-sliced indexes into sampling-based systems, as these two lines of works are orthogonal and can mutually enhance each other.

Ordering guarantee in AQP is studied by Kim *et al.* [20]. It proposes an optimal online sampling algorithm to guarantee the right order of groups with the help of efficient indexes.