Error-bounded Sampling for Analytics on Big Sparse Data

Ying Yan
Microsoft Research
ying.yan@microsoft.com

Liang Jeff Chen
Microsoft Research
jeche@microsoft.com

Zheng Zhang
Microsoft Research
zheng.zhang@microsoft.com

ABSTRACT

Aggregation queries are at the core of business intelligence and data analytics. In the big data era, many scalable shared-nothing systems have been developed to process aggregation queries over massive amount of data. Microsoft's SCOPE is a well-known instance in this category. Nevertheless, aggregation queries are still expensive, because query processing needs to consume the entire data set, which is often hundreds of terabytes. Data sampling is a technique that samples a small portion of data to process and returns an approximate result with an error bound, thereby reducing the query's execution time. While similar problems were studied in the database literature, we encountered new challenges that disable most of prior efforts: (1) error bounds are dictated by end users and cannot be compromised, (2) data is *sparse*, meaning data has a limited population but a wide range. For such cases, conventional uniform sampling often yield high sampling rates and thus deliver limited or no performance gains. In this paper, we propose error-bounded stratified sampling to reduce sample size. The technique relies on the insight that we may only reduce the sampling rate with the knowledge of data distributions. The technique has been implemented into Microsoft internal search query platform. Results show that the proposed approach can reduce up to 99% sample size comparing with uniform sampling, and its performance is robust against data volume and other key performance metrics.

1. INTRODUCTION

With the rapid growth in data volume, velocity and variety, efficient analysis of massive amount of data is attracting more and more attention in industry. In an Internet company, extracting and analyzing click streams or search logs help the company's mission critical businesses to improve service quality, find future revenue growth opportunities, monitor trends and detect root causes of live-site events in a timely fashion. At the core of these analytics tasks are decision support queries that aggregate massive amount of data. Nowadays, aggregation queries over big data are often processed in a shared-nothing distributed system that scales to thousands of machines. MapReduce and Hadoop are two well-known software frameworks. Microsoft's SCOPE is another instance in this category [4, 22]. It combines MapReduce's explicit parallel executions and SQL's declarative query constructs. Today inside Microsoft, SCOPE jobs are running on tens of thousands of machines everyday.

While a shared-nothing distributed system provides massive parallelism to improve performance, aggregation queries are still expensive to process. The fundamental cause is that query processing by default consumes the entire data set, which is often hundreds or thousands of terabytes. Our investigation in one cluster reveals that 90% of 2,000 data mining jobs are aggregation queries. These queries consume two-thousand machine hours on average, and some of them take up to 10 hours. They exhaust computation resources and block other time-sensitive jobs.

One technique to cope with the problem is sampling: queries are evaluated against a small randomly-sampled data, returning an approximated result with an error bound. An error bound is an interval in which the real value falls with a high possibility (a.k.a. confidence). Such an approximated result with an error bound is often good enough for analysis tasks. For instance, a study on a product's popularity along the time dimension may not care for the accurate sales number in a particular time, but only the trend. An analysis identifying the most promising sales region may be only interested in relative positions. A smaller amount of data to be processed improves not only query response time, but also the overall system throughput.

Data sampling for aggregation queries has been extensively studied in the database literature as *approximate query processing* [1, 3, 6, 10, 16]. The theory foundation lies in statistical sampling [11, 15]: given a uniform sample from a finite or infinite population, what is the estimation's error bound? Existing techniques utilize the theory and proceed with different system settings. For example, online aggregation [1] assumes that data is first shuffled and then processed in a pipelined execution, updating estimations and error bounds continuously. Query execution can be terminated whenever the error bound reaches a satisfactory level. Alternatively, a number of techniques use different sampling schemes to build data sketches offline and use them to answer queries at runtime.

When we tried to apply these known technologies in our production environment, however, we encounter a few critical challenges, summarized as below:

Proceedings of the VLDB Endowment, Vol. 7, No. 13

- **Tight error bounds**. Aggregation queries of data mining jobs in our environment are not standalone queries. Rather, they are often data providers for later consumers in the analytics pipeline. As such, the adoption of sampling is predicated on tight error bounds that must not be compromised.
- **Sparseness of data**. From the perspective of sampling, the sparseness of data depends on both the population and the value range. A medium or large population may still be deemed sparse by the theory of statistical sampling, if the value is distributed over a very wide range. In such cases, uniform sampling may require sample size as large as the raw data size in order to satisfy the error bound requirement. This brings no or negative performance gains, due to sampling overhead. The phenomena of sparse data are not uncommon in the production environment. Indeed, the 80/20 rule applies in many domains in practice. Figure 1 shows a typical data distribution over which an average needs to be computed; the pattern is long-tail. Uniform sampling with 20% error bound and 95% confidence needs to consume 99.91% of the data.

The unique combination of error bound requirements and sparse data invalidates most existing techniques. Conventional offline sampling schemes do not provide commitments to error bounds, but only promise to minimize them [1, 16, 3]. Other techniques, such as online aggregation [10] and bootstrapping [12], can provide an error bound guarantee. But they do not target sparse data intentionally. As a result, their executions often cannot stop early and have to process almost all data before the error bound requirement can be satisfied.

Intuitively, to deal with the above problems, error-bounded sampling must understand the data distribution, which is the central theme of this paper. This will translate into a more expensive way of generating samples. Such cost can be absorbed if samples can be shared among queries. Fortunately, this is the case in our production environment, as shown in Table 1. Among the 2,000 jobs we analyzed, 3 inputs (0.1%) are shared by more than 600 jobs and 37 inputs (1.2%) are shared by over 100 jobs. These jobs often share many commonalities on targeted attributes, if not completely the same. It means we can afford higher expenses on building samples of these inputs for the targeted attributes. Moreover, the cost of error-bounded sampling can further be amortized, if we can incrementally maintain samples upon updates. As we will discuss later, this is the most effective if/when data distributions change slowly over time.

Table 1: Inputs shared by multiple jobs

No. of Inputs	No. of Jobs Sharing
3	≥ 600
37	≥ 100
680	≥ 10
1700	≥ 2
580	0

1.1 Our Contribution

In this paper, we propose a new sampling scheme to address the aforementioned challenges. The technique is a variant of stratified sampling [13] and relies on the insight that we

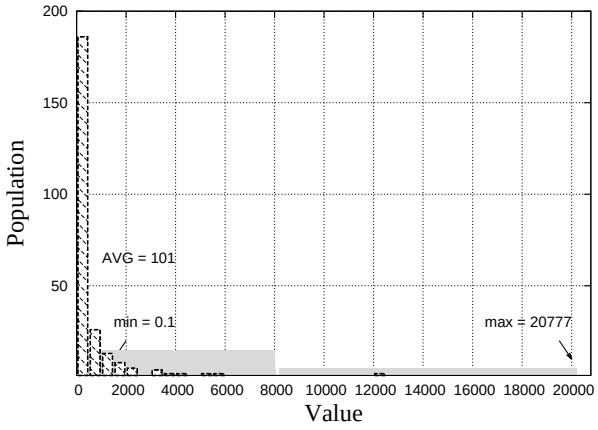


Figure 1: Sparseness of one representative production Data

can only reduce sampling rates and secure error bounds for sparse data if we have prior knowledge of its distribution. Consider the distribution in Figure 1, we can partition the data into two buckets, one covering the head and the other covering the rest. The small value range of the first bucket leads to a small sampling rate. As most of the data fall into the first bucket, the overall saving is substantial, even though (in the extreme case) all data in the second bucket must be included.

Bucket partitioning with bucket-dependent sampling rates is called a *sampling scheme*. Applying a sampling scheme to raw data generates a *sample sketch*. The total saving is achieved when we re-write eligible queries to run against the sketch, instead of the original data. Intuitively, deriving the best sampling scheme is a data-dependent optimization problem: given a complete view of the data and a user-specified error bound, find the sampling scheme such that the total sampled data is minimized. Unlike existing work that sets a space constraint, our formalization focuses on minimizing sample size while securing error bounds. Indeed, most application users we experienced are interested in result quality and response time, instead of space constraint.

The first contribution of this paper is identifying new requirements and challenges of data sampling for aggregation queries in the production environment. We present an algorithm for finding an optimal scheme of error-bounded sampling with the complexity of $O(N^4)$ (N being the number of data points), and a more practical heuristic algorithm with $O(N)$. The algorithm has an appealing property that its performance is no worse than the uniform sampling.

The most straightforward application of our technique is to build a sample sketch before queries are issued. As such, once data is updated, the sketch must be rebuilt, before it can answer new queries. This is a nuisance, as data is continuously updated in the product environment. Our second contribution is mechanisms that incrementally maintain sample sketches when recurring queries have temporal overlaps of inputs between successive invocations, e.g., landmark- or sliding-windowed queries. Incremental maintenance enables us to save computation resources for building sample sketches, without sacrificing error bound guarantees.

Our final contribution is a number of experiments we have run against real-world production jobs in SCOPE, with

and without our technique. These experiments have shown promising results: the saving is substantial and across the board, comparing to uniform sampling. For some production queries, our approach can save up to 99% of the sample size, or a thousand times of reduction. Moreover, our technique is robust against changes of data volume and other performance metrics. The lessons we learned here is also general: optimizations of big data computing demand a tighter coupling with data itself.

The remaining of the paper is organized as follows. Section 2 introduces preliminaries on data sampling for aggregation queries, and describes when and why existing techniques fail. Section 3 formalizes the problem and presents an algorithm that delivers the optimal solution, and the more practical heuristic algorithm. Error-bounded sketch updates are introduced in Section 4. Experimental results are reported in Section 5. Section 6 discusses related work. Section 7 concludes the paper.

2. PRELIMINARIES

We start by introducing sampling principles for aggregation queries. We limit ourself to queries with a single AVG aggregation in this and the next section, for ease of exposition of sampling principles and our new sampling schemes. We will extend the technique to a wider range of queries in Section 4.1.

An aggregation query Q specifies a set of GROUP-BY attributes and an aggregation attribute A over a source—either a structured table or a unstructured text file (e.g., a log file). Let $G = \{g_1, \ldots, g_m\}$ be the set of non-empty groups, and $X_g = \{x_1^g, \ldots, x_{N_g}^g\}$ be A's values in group g. The AVG value for group g, denoted by $\overline{X_g}$, is computed by aggregating the N_g values in g.

A sample over group g is a subset of n_g ($\leq N_g$) values. The estimated AVG value for group g based on this sample is:

$$\hat{X_g} = \frac{x_1^g + \ldots + x_{n_g}^g}{n_g}$$

and the estimation's error is:

$$\varepsilon_g = |\overline{X_g} - \hat{X_g}|$$

A sample is a probabilistic event. Two independent samples may yield different estimations and errors. We say that a sampling scheme for group g satisfies the error bound $\varepsilon_0 > 0$ under confidence $\delta \in [0, 1]$, iff among M independent samples produced by the scheme, at least $M \cdot \delta$ estimations satisfy $\varepsilon_g \leq \varepsilon_0$.

A sampling scheme for query Q is a union of sampling schemes for all groups G, so that all groups' estimations are close to their real values. It is not uncommon that users may only be interested in groups above/below a certain population. For instance, revenue analysis may focus on either top or bottom contributors, depending on application scenarios. In such cases, we may discard uninterested groups when designing a sampling scheme.

A common sampling scheme for a group of values is uniform sampling. That is: each value in group g is randomly kept or discarded with equal probability. A simple implementation is to generate a random number for each value to determine whether or not this value is kept. If we want to sample a fixed number of values from g, the reservoir sampling algorithm [18] can be used.

Given a confidence δ and sample size n_g, the Hoeffding equation [11] from statistical sampling defines the estimation's error bound:

$$\varepsilon_g = H(n_g, \delta) = (b_g - a_g)\sqrt{\frac{1}{2n_g} \ln \frac{2}{1-\delta}} \tag{1}$$

where a_g and b_g are lower and upper bounds of the values in group g. By this equation, given a user-specified error bound ε_0, we may compute how much data needs to be sampled so that the estimation's error bound is at most ε_0:

$$n_g = \min \begin{cases} \left\lceil \frac{(b_g - a_g)^2 \ln \frac{2}{1-\delta}}{2\varepsilon_0^2} \right\rceil \\ N_g \end{cases} \tag{2}$$

The Hoeffding equation originally targets sampling from an infinite population. Alternatively, it can be compensated by a factor for a finite population [15], yielding the sample size as:

$$n_g = \left\lceil \frac{1}{\frac{2\varepsilon_0^2}{(b_g - a_g)^2 \ln \frac{2}{1-\delta}} + \frac{1}{N_g}} \right\rceil \tag{3}$$

When the error bound ε_0 is large or $b_g - a_g$ is small, Equation 2 is equivalent to Equation 3. In other cases, Equation 3 provides a smaller sample size. Both equations provide theoretical guarantees on the error bound under the confidence. We will only be discussing Equation 1 (or Equation 2) in the following for simplicity. All the techniques and analysis presented later can also be applied by replacing Equation 2 with Equation 3.

Error-bounded uniform sampling scheme based on Equation 2 is straightforward. Its problem, however, is that when the values in group g spread over a wide range, the minimal data required to satisfy the error bound is close to the original data. Specifically, n_g is proportional to $(b_g - a_g)^2$ in Equation 2. When $b_g - a_g$ is large and N_g is not large enough, n_g/N_g will be close to 1. It means that, in order to satisfy the error bound, sampling can hardly get rid of any data; query execution over the sample would produce limited performance gains, or even penalties overall due to sampling overhead. In this paper, we refer to the data over a wide range (i.e., large $b_g - a_g$) but with a limited population (i.e., small N_g) as *sparse*. Even when data volume increases, sparseness problem still exists, especially when the number of GROUP-BY attributes increase.

3. ERROR-BOUNDED STRATIFIED SAMPLING

Our solution to big sparse data is based on the insight that we may only reduce the sample size and secure the error bound if we have better knowledge of X_g. Specifically, if we partition the range $[a_g, b_g]$ into buckets $[a_g^1, b_g^1] \cup \ldots \cup [a_g^k, b_g^k]$, we will be able to reduce sampling rates of individual buckets due to their smaller $(b_g^i - a_g^i)$ in Equation 1, thereby possibly reducing the total sample size too.

A simple sampling scheme based on the above intuition is to equally partition the data range and apply the Hoeffding equation to sub-ranges using the global error bound ε_0. While this simple scheme may reduce the sample size to some

extends, it ignores the data distribution and may not lead to a small sample. First, equal-length partitions mean that every sub-range is sampled in the same way, because $b_g^i - a_g^i$ and ε_0 are the same across all sub-ranges. But sub-ranges do not have to have the same error bound, as long as the global error bound requirement is satisfied. Intuitively, some sub-ranges are more important than others in influencing the global estimation and the error bound. This flexibility allows us to increase the error bounds for less critical sub-ranges and reduce the total sample size. Second, the number of sub-ranges is still a parameter that needs to be determined. More partitions do not necessarily lead to a smaller sample size. As we will discuss later, the sample size of a sub-range must be an integer. Due to integer rounding, the total sample size can only increase when we keep adding more partitions. In Figure 2, we compare equal-length partitioning and variable-length partitioning (using the algorithm introduced later). As we can see, even if we can find the optimal number of partitions, equal-length partitioning still produces a larger sample size than variable-length partitioning does.

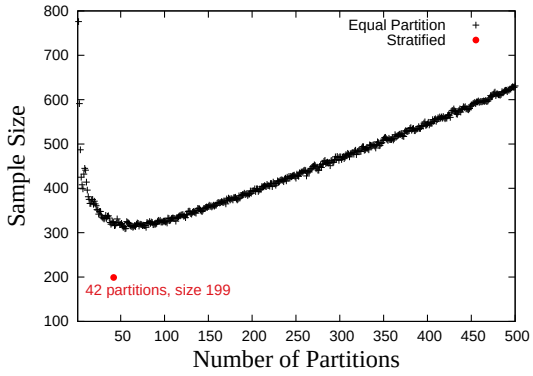


Figure 2: Equal-length partitioning v.s. variable-length Partitioning

We need an algorithm to find a sampling scheme that leads to an optimal sample size, while satisfying the global error bound requirement. In this section, we introduce error-bounded stratified sampling, a variant of stratified sampling [13], that aims to minimize the sample size while satisfying the error bound requirement. For ease of disposition, we only consider designing a sampling scheme for a single group of values $X = \{x_1, \ldots, x_N\}$ and omit all g's in sub- or super-scripts.

3.1 Problem Definition

Assume $X = \{x_1, \ldots, x_N\}, x_i \in [a, b]$ are sorted. A stratified sampling scheme specifies two types of parameters:

1. Bucket partitioning. The sampling scheme partitions the range into k buckets, $B_i = [a_i, b_i], i = 1, \ldots, k$. Bucket B_i contains N_i values.

2. Buckets' error bounds. Given bucket B_i's boundaries $[a_i, b_i]$ and its error bound ε_i, Equation 2 determines the number of values n_i must be sampled out of B_i's population N_i. A special case is when $\varepsilon_i = 0$, $n_i = N_i$, i.e. all data in the i-th bucket is kept.

Let $\hat{X}_i$ be the estimation using the n_i values in B_i. The total sample size is $n = n_1 + \ldots + n_k$. The global estimation $\hat{X}$ and the error bound ε_0 and those from individual buckets satisfy the following equations:

$$N = N_1 + \ldots + N_k \quad (4)$$

$$\hat{X} = \frac{N_1 \cdot \hat{X}_1 + \ldots + N_k \cdot \hat{X}_k}{N} \quad (5)$$

$$\varepsilon_0 = \frac{N_1 \cdot \varepsilon_1 + \ldots + N_k \cdot \varepsilon_k}{N} \quad (6)$$

PROBLEM 1 (ERROR-BOUNDED STRATIFIED SAMPLING). *Given N sorted values $\{x_1, \ldots, x_N\}, x_i \in [a, b]$, an error bound ε_0 and a confidence δ, find a partition of $[a, b]$ and the error bound ε_i for partition B_i, such that*

1. $\frac{N_1 \cdot \varepsilon_1 + \ldots + N_k \cdot \varepsilon_k}{N} \leq \varepsilon_0$, *and*

2. $n_1 + \ldots + n_k$ *is minimized.*

Note that in the problem definition, we do not limit the number of buckets k.

3.2 An Optimal Solution

We present a dynamic programming algorithm that gives an optimal solution to Problem 1. At the first glance, buckets' ranges and error bounds are parameters of a stratified sampling scheme that form the search space. An algorithm needs to search the space to find an optimal sampling scheme that yields the minimal sample size. However, error bounds from individual buckets may not be integers and we cannot enumerate them to search the whole space. Our algorithm is derived from an alternative view: while error bounds are innumerable, sample size can be enumerated. The algorithm enumerates the sample size from 1 through N (i.e., all data) and finds the minimal size that satisfies the error bound requirement.

Consider N sorted values $X = \{x_1, \ldots, x_N\}$. We define $\mathcal{E}_{\min}(x_i, m)$ to be the minimal error bound that any stratified sampling scheme can achieve when sampling m values from $[x_1, x_i]$ to estimate the AVG value in this range. Note that $[x_1, x_i]$ may consist of more than one bucket. When $m = 1$ or $m = i$, $\mathcal{E}_{\min}(x_i, m)$ is given by:

$$\mathcal{E}_{\min}(x_i, 1) = H(1, \delta) = (x_i - x_1)\sqrt{\frac{1}{2}\ln\frac{2}{1-\delta}} \quad (7)$$

$$\mathcal{E}_{\min}(x_i, i) = 0 \quad (8)$$

In Equation 7, only one value is sampled. The error bound is given by the Hoeffding equation with sample size set to 1. In Equation 8, the minimal error bound is 0, because all data are processed and the estimation is accurate.

When m is between 1 and i, $\mathcal{E}_{\min}(x_i, m)$ is computed from one of the two cases: (1) the m values are sampled uniformly from $[x_1, x_i]$, or (2) they are sampled from two or more buckets with different sampling rates. For the first case, $\mathcal{E}_{\min}(x_i, m)$ is given by the Hoeffding equation.

For the second case, we decompose the computation of $\mathcal{E}_{\min}(x_i, m)$ into two components: let $[x_{c+1}, x_i]$ be the rightmost bucket of $[x_1, x_i]$, and $[x_1, x_c]$ be the remaining sub-range. We designate s values sampled from the sub-range, and the remaining $m - s$ values *uniformly* sampled from the rightmost bucket. The global error bound of the m values

from $[x_1, x_i]$ is given by Equation 6 that combines the two sub-ranges' error bounds:

$$\frac{\mathcal{E}_{\min}(x_c, s) \cdot c + H(m-s, \delta) \cdot (i-c)}{i}$$

By the definition of $\mathcal{E}_{\min}(x_i, m)$, we need to find c and s that lead to the minimal error bound of all possible rightmost buckets when sampling m valuse.

Combining the two cases, we have:

$$\mathcal{E}_{\min}(x_i, m) = \min \begin{cases} \mathcal{E}^d_{\min}(x_i, m) \\ (x_i - x_1)\sqrt{\frac{1}{2m} \ln \frac{2}{1-\delta}} \end{cases} \quad (9)$$

$$\mathcal{E}^d_{min}(x_i, m) = \underset{\substack{1 \le c < i \\ 1 \le s \le \min\{c, m-1\}}}{\arg\min} \Big(\mathcal{E}_{\min}(x_c, s) \cdot c + (x_i - x_{c+1}) \sqrt{\frac{1}{2(m-s)} \ln \frac{2}{1-\delta}} \cdot (i-c) \Big) \Big/ i$$

By Equation 9, we can compute $\mathcal{E}_{\min}(x_N, 1)$ through $\mathcal{E}_{\min}(x_N, N)$. The optimal solution to Problem 1 is the minimal m such that $\mathcal{E}_{\min}(x_N, m) \le \varepsilon_0$. Backtracking m_{min} will give us buckets' ranges and sampling rates.

THEOREM 1. *The complexity of the algorithm based on Equation 9 is $O(N^4)$.*

To compute $\mathcal{E}_{\min}(x_N, 1)$ through $\mathcal{E}_{\min}(x_N, N)$, there are $O(N^2)$ entries to fill. To compute each entry, the min operator in Equation 9 iterates through c and s, and compares $O(N^2)$ values. The final step on finding m_{min} is $O(N)$. Altogether, the algorithm's complexity is $O(N^4)$.

3.3 Heuristic Algorithms

The algorithm presented in last section has a high complexity. Our experience with SCOPE is that any sampling algorithm whose complexity is higher than $O(N^2)$ will introduce considerable cost that outweighs sampling's benefits. In this section, we present a heuristic algorithm that compromises the optimality but improves efficiency. We show that while the algorithm may not lead to the minimal sampling size, the derived stratified sampling cannot be worse than conventional uniform sampling and often lead to significant improvements.

One cause of the algorithm's high complexity is variable error bounds of individual buckets. A simplification is that we limit each bucket's error bound to be the same, i.e., $\varepsilon_1 = ... = \varepsilon_k = \varepsilon_0$. By Equation 6, the global error bound requirement is obviously satisfied.

When we set $\varepsilon_i = \varepsilon_0, i \in [1, k]$, an $O(N^2)$ algorithm can find the global optimum: let $S(x_i)$ be the minimal sample size between $[x_1, x_i]$ while each bucket's error bound is ε_0. A dynamic programming algorithm that finds the minimal sample size is illustrated in the following equation:

$$S(x_i) = \underset{1 \le j \le i-1}{\arg\min} \left(S(x_{j-1}) + \min \left\{ \frac{(x_j - x_i)^2 \ln \frac{2}{1-\delta}}{2\varepsilon_0^2}, N_i - N_j \right\} \right)$$

Instead of seeking the global optimum, we use a local-optimum search heuristic to further reduce complexity. The pseudo code is shown in Algorithm 1. Given an array of sorted values, the algorithm scans the array from the head to the tail. For each new value encountered, the algorithm tries to merge it into the current bucket (Line 5). If the merging extends the bucket's value range and increases the number of values to be sampled under the error bound ε_0, a new bucket is created (Line 10 through 12); otherwise, the value is merged into the bucket and the bucket's range is updated (Line 8). The algorithm's complexity is $O(N)$.

Algorithm 1: A linear algorithm for finding an error-bounded stratified sampling scheme

Input: An error bound ε_0, a sorted array of values $X = \{x_1, \ldots, x_N\}$
Output: A set of buckets $\mathcal{B} = \{B_1, \ldots, B_k\}$

```
1:  B_curr = [x_1, x_1], B_curr.SampleSize = 1
2:  B = {B_curr}
3:  i ← 2
4:  while i ≤ N do
5:    Let B_t = [B_curr.lower, x_i]
6:    Compute B_t.SampleSize by Equation 2
7:    if B_curr.SampleSize + 1 ≤ B_t.SampleSize then
8:      B_curr = B_t
9:    else
10:     B_new = [x_i, x_i], B_new.SampleSize = 1
11:     B ← B ∪ {B_new}
12:     B_curr = B_new
13:   end if
14:   i ← i + 1
15: end while
16: return B
```

THEOREM 2. *Given a user-specified error bound ε_0, the stratified sampling scheme given by Algorithm 1 samples no more data than uniform sampling.*

PROOF. If the algorithm returns one bucket, stratified sampling is equivalent to uniform sampling. Next, we prove that stratified sampling with two buckets samples no more data than uniform sampling.

Let $[x_1, x_N]$ be the value range, and $[x_1, x_c]$ and $[x_{c+1}, x_N]$ be the two buckets' ranges. From the Hoeffding equation, we have:

$$\begin{aligned} & \ln \frac{2}{1-\delta} \cdot \frac{(x_N - x_1)^2}{2\varepsilon^2} \\ \ge \; & \ln \frac{2}{1-\delta} \left[\frac{(x_N - x_{c+1})^2}{2\varepsilon^2} + \frac{(x_c - x_1)^2}{2\varepsilon^2} \right] \end{aligned} \quad (10)$$

By case analysis, we can also prove the following inequality:

$$\min\{x + y, N\} \ge \min\{x, N_1\} + \min\{y, N_2\} \quad (11)$$

where $N = N_1 + N_2$. Putting Equation 10 and 11 together, we have:

$$\begin{aligned} & \min \left\{ \ln \frac{2}{1-\delta} \cdot \frac{(x_N - x_1)^2}{2\varepsilon^2}, N \right\} \\ \ge \; & \min \left\{ \ln \frac{2}{1-\delta} \frac{(x_N - x_{c+1})^2}{2\varepsilon^2}, N_1 \right\} + \\ & \min \left\{ \ln \frac{2}{1-\delta} \frac{(x_c - x_1)^2}{2\varepsilon^2}, N_2 \right\} \end{aligned}$$

The left hand of the inequality is the sample size when applying uniform sampling. The right hand is the total sample size when using uniform sampling for two buckets.

For stratified samplings with more than two buckets, we can prove by induction: let $[x_j, x_N]$ be the rightmost bucket and $[x_1, x_{j-1}]$ be the remaining range. We first show that using two or more buckets for $[x_1, x_{j-1}]$ is no worse than using one bucket. Then we show that combining $[x_1, x_{j-1}]$ and $[x_j, x_N]$ is no worse than uniform sampling over $[x_1, x_N]$. □

Even when the data is uniformly distributed, stratified sampling can still drive down the sample size through range partitioning, because inequality functions 10 and 11 are distribution-oblivious. However, they do not always imply "the more buckets the better" claim. Intuitively, when $x_N - x_1$ is small enough, adding more buckets would not help. In particular, the sample size computed by the Hoeffding equation must be rounded to integers. When adding buckets only changes decimals, there is no need to do so.

4. OFFLINE SAMPLING

This section develops necessary steps and extensions in order to apply the core algorithm to real-world production workloads. We first extend single-AVG queries to a wider scope of queries. We then discuss generating sampling schemes offline, which is then used to build sample sketches to answer queries. Finally, we study the problem of incremental maintenance of sample sketches in the face of changing data, and present algorithms for two most common update modes.

4.1 Beyond a Single AVG Aggregation

So far we have been focusing on queries with a single AVG aggregation. We extend the proposed sampling technique in two ways: queries with SUM and COUNT aggregations, and queries with multiple aggregations.

A COUNT aggregation calculates the number of records from an input. Unlike the average, this number is proportional to input size, a parameter that cannot be derived from a sample but only through a full scan. Fortunately, this number is usually computed once when a sample is built. It is later stored as meta-data and does not need online computations any more.

Given an AVG estimation and the data size, a SUM aggregation can be easily computed by multiplying the average and the full data size. The SUM estimation's confidence is the same as that of AVG. Its error bound is $\varepsilon_{avg} \cdot N$ where ε_{avg} is the error bound of the AVG estimation and N is the data size. In general, any aggregation that is an algebraic combination f of basic aggregations (i.e., SUM, AVG or COUNT) can be estimated by combining their individual estimations $f(e_1, \ldots, e_n)$. The error bound is given by the minimal and the maximal values $f(e_1 \pm \varepsilon_1, \ldots, e_n \pm \varepsilon_n)$ can achieve.

When a query includes more than one aggregation (over more than one column), we may decompose the original query into multiple sub-queries, and generate one sample for each aggregation. This approach requires efforts to decompose the input query and the subsequent step to join sub-queries' results. Alternatively, we may produce one sample that serves all aggregations. This can be done by union sample schemes of each individual columns. The combined scheme is essentially a voting machine: a row or record from the input is sampled if any scheme says so. While this mechanism may over-sample in the sense that a sample may not be needed for all aggregations, it ensures that all aggregations' estimations can satisfy the error bound requirements.

4.2 Sampling Scheme and Sample Sketches

The output of a sampling algorithm for a group of values is a *sampling scheme*. A sampling scheme is a set of buckets $\mathcal{B} = \{B_1, \ldots, B_k\}$, each represented as a triple $B_i = \langle l_i, u_i, \varepsilon_i \rangle$. ε_i is B_i's error bound, and determines how many records B_i is expect to receive, as specified by Equation 2. We use ε_i instead of ε_0 or n_i to deal with the case when we need to reuse the sampling schemes over changing data, as will be clear later. Also, if $\varepsilon_i = 0$, B_i's sample size n_i is as large as its population N_i. When there are multiple groups $g_1, \ldots, g_n$, the sampling scheme is the union of the sampling schemes of all groups: $\{\mathcal{B}^{g_1}, \ldots, \mathcal{B}^{g_n}\}$.

We can use a sampling scheme to sample raw data and produce a *sample sketch*. A sample sketch is basically a subset of raw data. In addition to raw data's columns, each record in the sketch contains one additional column ScaleFactor. The value of this column is $N_j^{g_i}/n_j^{g_i}$, the inverse of the overall sampling rate of bucket $B_j^{g_i}$ to which the record belongs. This column is maintained to facilitate estimating aggregations that are scale-dependent, e.g., SUM, so that during query answering we can reconstruct the estimation of the real scale. A similar technique is introduced in [1].

To answer queries using sketches, we need to rewrite queries by replacing the input with the sample sketch. Scale-independent aggregation operators, such as AVG, are unchanged; scale-dependent aggregation operators, such as SUM and COUNT, need to incorporate ScaleFactor to reconstruct the estimation for the raw data's scale. Similar to the approach introduced in [1], a SUM aggregation SUM(X) is rewritten to SUM(X*ScaleFactor); a COUNT aggregation COUNT(*) is rewritten to SUM(ScaleFactor).

Given a sampling scheme, a sketch building processor computes a sample sketch by scanning the raw data. For each record/row scanned, the processor finds the group g_i and the bucket $B_j^{g_i}$ that this record falls in. The processor then determines whether this record should be kept or discarded, according to its error bound $\varepsilon_j^{g_i}$: $\varepsilon_j^{g_i}$ specifies how many records (i.e., $n_j^{g_i}$) must be kept in bucket $B_j^{g_i}$. This is implemented by inserting records into a reservoir and randomly replacing records in the reservoir after it is full. This is also known as the reservoir sampling algorithm [18]. After all data is scanned, bucket $B_j^{g_i}$'s population $N_j^{g_i}$ is also known, from which ScaleFactor can be computed.

Overall, building a sample sketch consists of two steps: generating a sampling scheme and building a sample sketch. These two steps need to scan raw data twice, incurring non-negligible cost. However, this generated sample sketch can be used by multiple queries many times, effectively amortizing the cost to a low level.

4.3 Incremental Maintenance of Sample Sketches

The sampling algorithms proposed in Section 3.3 all have complexities no lower than $O(N)$, which can still be deemed expensive when the data scale is very large. While a sample sketch is only built once and can be re-used by many queries, re-building a new sketch from scratch upon data updates will incur significant overhead, especially when updates are frequent.

A common approach to circumvent this overhead is to incrementally update the sample sketch, so that each time only updated data is processed. This approach fits our applications well, because our production jobs mainly target logs

that continuously receive new data on an hourly or daily basis. We are particularly interested in two update modes that are widely observed in production jobs: *landmark windowed* queries and *sliding windowed* queries. Landmark windowed queries target data from a landmark to present.When new logs are appended, they are patched to the sample sketch. Sliding windowed queries focus on logs in a fixed-length window. As new logs arrive, out-of-interest old logs are removed from the sketch. In either cases, logs are immutable, and the queries only see log additions and deletions.

In the following, we first analyze the maintenance guideline and then discuss detailed maintenance mechanisms of the two modes.

4.3.1 Error-bounded Maintenance

A sampling scheme of a group of values is represented as $\mathcal{B} = \{B_1, \ldots, B_k\}$ and $B_i = \langle l_i, u_i, \varepsilon_i \rangle$. When new data is added or old data is deleted, samples in individual buckets are changed too. Let B_i's new sample size and population be n_i' and N_i' after updates. B_i's new error bound ε_i' is computed by Equation 12:

$$\varepsilon_i' = \begin{cases} (b_i - a_i)\sqrt{\frac{1}{2n_i'}\ln\frac{2}{1-\delta}} & n_i' < N_i' \\ 0 & n_i' = N_i' \end{cases} \quad (12)$$

Incremental maintenance algorithms must ensure that the global error bound ε' is still no bigger than the initial error bound requirement ε_0 after the updates, i.e.,

$$\varepsilon' = \frac{N_1' \cdot \varepsilon_1' + \ldots + N_k' \cdot \varepsilon_k'}{N_1' + \ldots + N_k'} \leq \varepsilon_0 \quad (13)$$

When the above equation cannot be satisfied, the sample sketch cannot be incrementally updated; a new sample sketch must be re-built from scratch.

Remarks Incrementally updating a sample sketch is equivalent to using the old sampling scheme to sample the new snapshot. While by Equation 13 we can ensure the error bound requirement is still satisfied, the updated sample's size may not be optimal, because the new snapshot may describe a new distribution, demanding a new sampling scheme to achieve optimality. Nevertheless, incremental maintenance is still appealing to our applications. Logs received from various sources often reflect applications' states or users' behaviors. These signals usually change gradually, and so do the underlying data distributions. A sampling system using an old scheme can still keep the sample size at a low level, while benefiting from the low maintenance overhead. From an engineering perspective, we may periodically re-compute the sampling scheme, e.g., once in a month, and use the same scheme to incrementally maintain the sample sketch during the time window.

4.3.2 Landmark Windowed Queries

A landmark windowed query is over the data from a landmark to present. Once a sample sketch is built over an old snapshot, newly added data is incrementally patched to the sketch. In other words, the maintenance is insertion only.

Maintaining a sample sketch upon data insertions is similar to the sample building procedure. When a new data record x_i arrives, the maintenance algorithm inserts it to the sketch with a certain probability. Specifically, the algorithm first looks up bucket B_i to which this record belongs, and increases B_i's population to $N_i' = N_i + 1$. With reservoir sampling, this new record randomly replaces an existing record in B_i. Since B_i's population is changed to N_i', the ScaleFactor column of all the records in the bucket are updated as well.

A special case during the maintenance is that a new record x_i may fall out of all existing buckets. It means that this new value is out of the boundaries of the prior data distribution. Such new records must all be kept in the sample sketch, with ScaleFactor $= 1$. Intuitively, we have no knowledge of the data distribution out of existing boundaries. The only way we are able to secure the error bound is that we sample all of them. It is equivalent to adding a new bucket $B_{x_i} = \langle x_i, x_i, \varepsilon = 0 \rangle$ to the sampling scheme $\mathcal{B}$.

By using the above maintenance algorithm, the error bound in the updated sample sketch is always no bigger than the initial error bound. This is because the error bounds of existing buckets never change and the error bounds of new buckets are always zero. Overall, error bound is still guaranteed.

4.3.3 Sliding Windowed Queries

A sliding windowed query targets data in a fixed-length window. As new data arrives, it is patched to the sample sketch. Old data out of interests is removed from the raw data and the sketch. Let $\mathcal{B} = \{B_1, \ldots, B_k\}$ be a group's sampling scheme. We also use B_i to denote a set of sampled records in this bucket. Let N_i^d be the number of old records in the raw data, and n_i^d be the number of old records in bucket B_i in the sketch. Let N_i^a be the number of new incoming records in the raw data. While all n_i^d old records must be removed from B_i, only a fraction of N_i^a records may be eventually added to the sketch, because not all new data has to be sampled. We denote this number as $n_i^a \leq N_i^a$.

After data is removed and added to the sketch, a bucket B_i's population becomes

$$N_i' = N_i - N_i^d + N_i^a$$

The number of sampled records in B_i is

$$n_i' = n_i - n_i^d + n_i^a$$

Given N_i', n_i' and B_i's boundaries $[a_i, b_i]$, Equation 12 gives B_i's new error bound. The new sketch must satisfy the global error bound, as specified by Equation 13.

Among all the above variables, N_i^d, N_i^a and n_i^d are known. A maintenance algorithm needs to find n_i^a, so that each bucket knows its reservoir size. This is an integer programming problem: the algorithm aims to find $n_1^a, \ldots, n_k^a$, such that the size of the updated sample sketch is minimized:

$$\begin{array}{ll} \min & \sum_{i=1}^k (n_i - n_i^d + n_i^a) \\ \text{subject to} & \frac{\sum_{i=1}^k N_i' \cdot G(N_i', n_i', a_i, b_i, \delta)}{\sum_{i=1}^k N_i'} \leq \varepsilon_0 \\ & 0 \leq n_i^a \leq N_i^a \\ & n_i^a \in \mathbb{Z} \end{array}$$

where $G(\cdot)$ is a shorthand for Equation 12. Note that n_i^a has a finite space, from 0 to N_i^a. It is possible that any value in the space cannot satisfy the global error bound requirement. In such a case, the sketch cannot be incrementally maintained, and must be re-built from scratch.

Integer programming is known to be NP-hard. We propose a heuristic algorithm (see Appendix) to find a maintenance scheme for the sketch. The idea is to consider buckets with high populations first. A record added to these buckets

is likely to have more impacts in reducing the global error bound than the one added to low-population buckets, because the global error is a weighted sum. The algorithm iterates through all buckets, from the highest population to the lowest one, and uses a local optimal heuristic in deciding how much data to be added to each bucket.

5. EXPERIMENTAL STUDY

In this section, we report an experimental study of error-bounded sampling. We implemented the proposed techniques in SCOPE and evaluated their performance. All experiments were run in a cluster of 200 nodes. To our best knowledge, no existing techniques target big sparse data with error bound guarantees. Hence, we compare our approaches against error-bounded uniform sampling that randomly samples data in each group. The sample size is given by Equation 2 for each group. This is also the approach used in a recent system that promises error bounds [2]. In the following figures, we use Stratified to denote our approach and Uniform to denote the uniform sampling.

We drive the experiments with both synthetic and production workloads. For the latter, we use log streams that consist of various search logs whose sizes are 20 TB on average. We select 50 representative queries out of hundreds of daily jobs. These queries are in the following template:

```
SELECT    C1, C2 ... Cn,
          AggFunc(A1), AggFunc(A2)... AggFunc(Am)
FROM      LogStream PARAMS(StartDate, EndDate)
WHERE     Predicates
GROUP BY C1, C2 ...Cn;
```

The number of GROUP-BY attributes n varies from 2 to 5, and the number of aggregations m ranges from 1 to 3. The aggregation operators are SUM, COUNT and AVG. These queries probe different base logs, ranging from one day to one month. The selectivities of the queries' predicates vary from 0.01% to 80%.

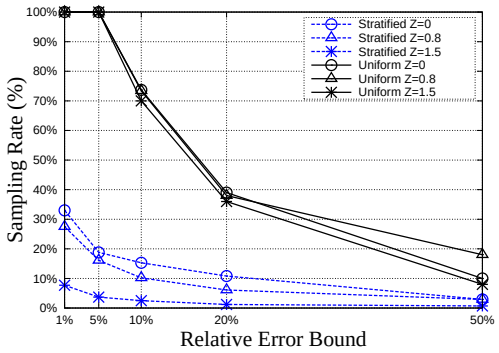
All errors in the experiments are reported as relative errors, to help readers understand approximation introduced by sampling without revealing application contexts.

5.1 Effects of Data Skewness

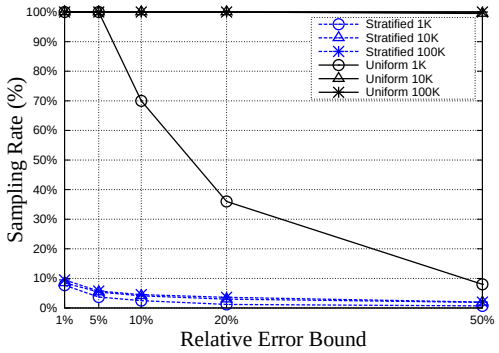
Sparse data is the major factor that invalidates most prior efforts. In this subsection, we evaluate the effectiveness of our sampling technique over sparse data. We first use synthetic data sets whose sparseness is quantifiable. We then randomly select 10 production queries to show the real sparseness of production data and examine the effectiveness of our approach.

To quantify data sparseness, we use the Zipf distribution whose skewness can be controlled by the z-parameter. When z is 0, the data is uniformly distributed; When z is 1.5, data is highly skewed (and sparse by our definition). We generate a number of single-group data sets whose values range from 1K to 100K. Each data set has the same size: 10K. For a fair comparison, we fix the mean to 500. The aggregation operator is AVG.

First, we examine the effects of data skewness under different error bound constraints. We fix the value range to 1K, confidence to 95% and vary the error bound and data skewness. As shown in Figure 3(a), our approach generates much smaller samples than the uniform sampling under different error bounds. For the uniform sampling, when the



(a) Sampling rate with data skewness



(b) Sampling rate with value range

Figure 3: Effects of data sparseness

error bound is small, even if data is uniformly distributed ($z = 0$), the sampling rate is still close to 100%. When data is more skewed ($z = 0.8$ and 1.5), uniform sampling shows similar performance. The reason is that uniform sampling is distribution-oblivious; the Hoeffding equation is based on the value range and the error bound, which in this case yield a very high sampling rate. By comparison, our approach is distribution-aware. It creates buckets by data distribution. When data becomes more skewed, some buckets have higher populations and smaller sampling rates. Though we may have to use a high sampling rate for the remaining buckets, their populations are not significant. Therefore, the total sample size is small.

Second, we evaluate sampling performance with different value ranges. We change the value range from 1K to 100K and fix z to be 1.5. The sampling rates of both approaches are compared in Figure 3(b). When the value range increases, by Equation 2, the sample size increases. Therefore, the uniform sampling generates larger samples. When the value range increases to 10K, the uniform sampling samples almost 100% of data even when the error bound reaches 20%. When the value range becomes 100K, it almost needs to sample the whole data set to satisfy the 50% error bound. By comparison, our stratified approach always generate small samples with different value ranges.

Next, we inspect the impact of confidence by varying it from 50% to 95%, with a fixed value range 1K and an error bound 5%. As illustrated in Figure 4, as confidence increases, the sample size of the uniform sampling increases dramatically However, our approach shows stable performance. When the confidence becomes 95%, the uniform sampling needs to sample 7 times more data than the stratified approach does in order to satisfy a 5% error bound.

From the above experiments, we conclude that our stratified sampling performs much better under different data sparseness, error bound and confidence settings than the baseline of the uniform sampling.

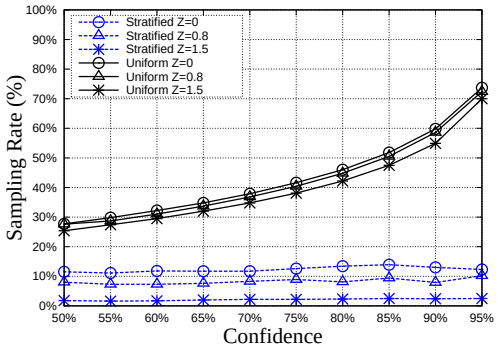


Figure 4: Effects of confidence

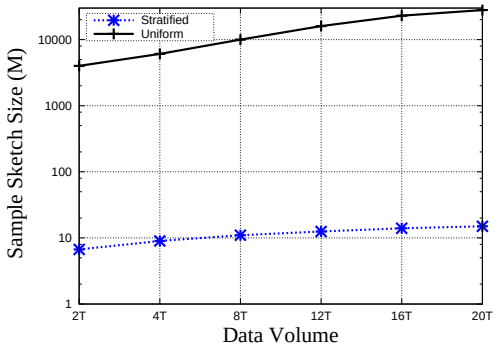
Finally, we evaluate both approaches for real-life production workloads. We randomly select 10 production queries and generate the offline samples using the uniform sampling and the stratified sampling. The error bound is 5% and confidence is 95%. The input data volume of those queries is 8T on average. Since some of the queries' predicates are selective, we cannot calculate the sampling rate by using the original data size as the base. Instead, we use the size of the data after the predicates' evaluation as the base. The results are illustrated in Table 2. We can see that when using the uniform sampling, the sampling rates range from 57% to 97%. The stratified sampling, on the other hand, is able to reduce the sampling rate to less than 5%. For Query 4, the stratified sampling rate is only 0.06%, saving 99% more compared with the uniform sampling. This query has 5 groups and most of them are big sparse data, and thus stratified sampling is particularly effective.

5.2 Effects of Data Volume

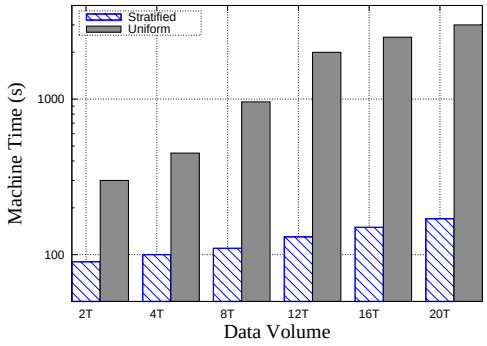
In this section, we evaluate the performance of both approaches over the inputs with different data volumes. The experiments are conducted using production workloads. We fix the error bound to 5%, confidence to 95% and build offline sample sketches over log streams whose sizes range from 2T to 20T. The performance results we show here are the average of a number of selected queries.

First, we examine the effects of data volume on the sample size. As shown in Figure 5(a), as the input volume increases, both approaches generate larger samples. For the uniform sampling, the sample size grows linearly with respect to data volume. This is because when data volume increases, the value range increases too. By Equation 2, the sample size increases as the value range increases. For the stratified approach, the increase of sample size is not dramatic, because it has resilient performance against different value ranges as shown in the experiments in Section 5.1. When the input log grows to be 20 TB, the uniform sampling produces 1340 times more data than the stratified approach.

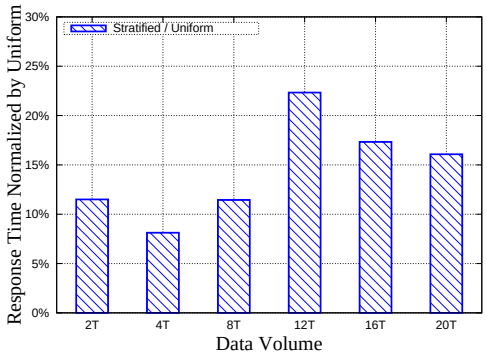
In addition to sample sizes, machine time is a measure that sums up the processing time of all participating nodes in SCOPE. It reflects how many computation resources a query consume. In Figure 5(b), we use the samples built from two approaches to answer queries and report machine time of query processing. As we can see, the machine time



(a) Sample size over different data volumes



(b) Machine time over different data volumes



(c) Response time over different data volumes

Figure 5: Effects of data volume

increases linearly with the increase of data size for the uniform sampling. This is due to the increase of sample size. Our stratified approach consumes much less machine time, because it generates smaller samples for queries to use. A careful reader may notice that while the samples built from the uniform sampling are a few hundred times larger than those built from the stratified sampling, the machine time of using the former samples is only dozens of times bigger. The reason is that there is a relatively bigger initialization cost in the early stages of the query execution pipeline, a cost that is independent of data volume.

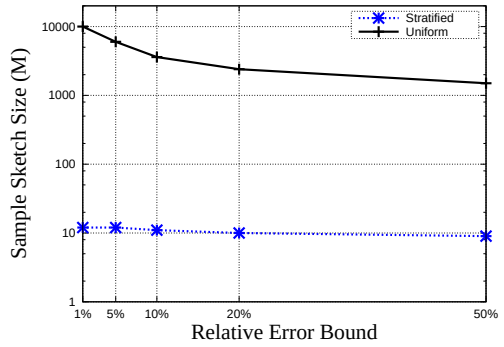
We also measure query execution time in this experiment. In a parallel environment, query execution time is dependent on how many machines or nodes are allocated for a query. In SCOPE scheduling, the number of nodes allocated to a query is usually determined by the size of the input. Such scheduling means that it is not completely fair to compare query execution time when we use samples generated by different approaches for query answering, because these samples may have very different sizes. At any rate, we normalize

Table 2: Sampling rate of production workload

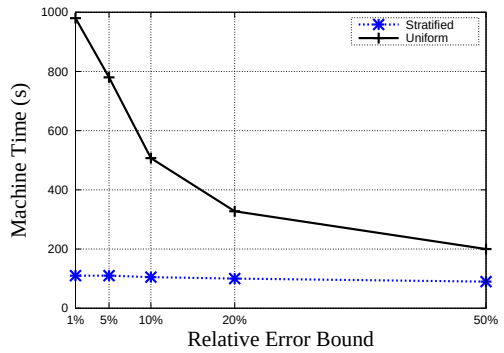
Sampling Approach	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
Uniform	89.85%	75.37%	94.38%	57.11%	81.81%	85.89%	92.64%	96.96%	95.07%	86.71%
Stratified	0.17%	0.14%	0.11%	0.06%	0.1%	4.86%	3.04%	1.5%	2.01%	0.11%

query execution time of the stratified sampling by that of the uniform sampling and show the results in Figure 5(c). We can see that the stratified sampling takes only 7% to 23% of the execution time of the uniform sampling. Savings in query execution time is not as significant as savings in machine time, since these queries are scheduled with fewer nodes because their inputs (i.e. the samples) are smaller. This means that the our sampling approach saves both execution time and resources.

5.3 Effects of Error Bounds



(a) Sample size with different error bounds



(b) Machine time with different error bounds

Figure 6: Effects of error bound

We compare both approaches with different error bounds in this group of experiments, using production workloads. Results here are averaged among all selected queries. The confidence is set to 95%. The input stream is 8 TB. The sizes of the samples produced by the two approaches are shown in Figure 6(a). As we can see, when the error bound increases, the sample size of the uniform sampling decreases dramatically. Stratified approach is able to produce a small number of samples with different error bound constraints.

The machine time of using samples to answer queries over different data volumes is shown in Figure 6(b). For both approaches, machine time drops with respect to the increases of error bounds. Comparing with the uniform sampling, the stratified sampling is much more robust against the change of error bounds. The machine time saved by the stratified sampling compared with the uniform sampling is not as significant as sample sizes because of the query initialization cost in SCOPE.

5.4 Incremental Maintenance of Samples

In this group of experiments, we evaluate the performance of the algorithms that incrementally maintain samples for sliding-windowed queries, by varying window sizes, sliding steps and error bounds over one-week production log stream with production workloads. Results here are averaged among all selected queries. First, we set the sliding step to 3 hours, window size to 2 days and change the error bound. The sample sizes through either incremental maintenance or rebuilding are shown in Figure 7. The figure shows that in most cases incrementally updated samples are not significantly larger than the rebuilt ones, if maintenance does not violate the error bound guarantee. In the worse-case scenario, the updated sample is 50% larger than the one rebuilt from scratch. This increase is significant. But given that the sample size is itself very small, such an increase is acceptable in practice.

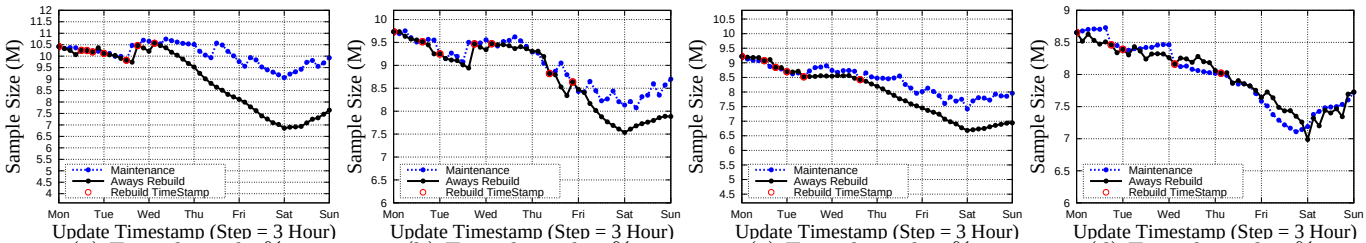
An interesting observation of Figure 7 is that sometimes the sample size after incremental maintenance can even be smaller than rebuilt samples, e.g., as shown in Figure 7(d). The explanation is that the heuristic stratified algorithm assumes that the error bound of each bucket is the same, in order to lower the algorithm's complexity. During incremental maintenance when old data is deleted and new data is inserted, the maintenance algorithm may adjust the buckets' error bounds such that the global error bound requirement is satisfied. It means that some buckets have error bounds higher than the default one, and therefore produce smaller samples than the samples before the maintenance. This is also the reason why we maintain error bound instead of sample size in the sampling scheme.

During incremental maintenance, it is possible that a sample after maintenance cannot satisfy the error bound constraint and therefore it must be rebuilt. These cases are highlighted in Figure 7 as circled points. Across the four figures in Figure 7, the number of rebuilding timestamps reduces, as the error bound increases. This is understandable: the larger the error bound, the more space the algorithm has to adjust each bucket's error bound and therefore make the global error bound smaller. Overall, the number of rebuilding timestamps is not significant for real-life workloads and data; samples can be incrementally updated in many cases, saving a fair amount of computation resources.

In the second experiment in this group, we set the error bound to 15%, sliding step to 12 hours and change the window size from 2 days to 4 days. We report the results in Figure 8. Similar to last experiment, the sample size after incremental maintenance is close to that of the rebuilt sample, except a handful of cases when the sample must be rebuilt.

6. RELATED WORK

Data sampling for aggregation queries has been studied in the database literature as approximate query processing



(a) Error bound 5% (b) Error bound 10% (c) Error bound 15% (d) Error bound 20%

Figure 7: Incremental maintenance with different error bounds

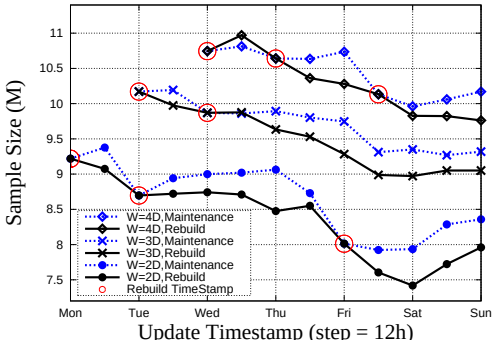


Figure 8: Incremental maintenance with different sliding window sizes

(AQP). Chaudhuri et al. gave a thorough survey in AQP [6]. According to the optimization goals, sampling techniques can be classified into two categories: (1) Space constraint—minimizing the error bound within limited sample space, and (2) error bound constraint—minimizing the sample size while satisfying a pre-defined error bound, which is the problem we addressed in this paper.

For the first category, the maximum available sampling space (or sample size) is considered as a budget constraint, a setting that is common in traditional database management systems. The optimization target is to minimize the error bound. A few papers presented biased sampling approaches for GROUP-BY queries when data distributions across groups have big variances [1, 16]. Chaudhuri et al. proposed to build the outliers of data into a separate index [5]. This approach, however, is only effective when data skewness is caused by outliers or deviants. Chaudhuri et al. also proposed a stratified sampling plan for lifted workload to minimize the errors [6]. Ganti et al. proposed a weight-based approach that prefers to sample data records that are asked by most queries [8]. As a result, this approach will deliver poor error estimations for less frequent queries, which is unacceptable in many scenarios. Babcock et al. built a family of precomputed samples to answer queries [3]. For each incoming query, a subset of samples is selected dynamically to improve the accuracy. Overall, the techniques in the first category can not be applied directly to solve the error bounded problem that we are addressing.

Techniques in the second category can provide guarantees on the error bound. Online aggregation [10] and its variants[20, 21, 19, 9] process users' aggregation queries in an online fashion. It continuously produces aggregated results together with an error bound and a confidence. Users can stop the execution whenever the error bound meets their requirement. Some efforts have been focused on implementing online aggregation in MapReduce environments [7, 14]. SciBORQ is a new system that restricts error bounds for analytical queries [17]. Recent attempts use the bootstrapping technique over the MapReduce framework [12]. The idea is to perform multiple sampling iterations until the computed errors are below the user's requirements. Nevertheless, these "on-the-fly" approaches do not target sparse data intentionally. As a result, their executions often cannot stop early and have to process almost all data before the error bound can be satisfied. In the recent work BlinkDB [2], the authors build multiple offline samples with different error bounds. In the online stage, they select the most appropriate sample to answer a query. However, they apply stratified sampling on individual groups, which can not be effective when data is sparse inside each group. As such, these techniques may not be effective.

The error-bounded stratified sampling technique proposed in this paper not only solves the error-bound-constrained problem, but also is effective when data is sparse, a common setting for in-production workloads.

7. CONCLUSIONS

Sampling as a means to reduce the cost of aggregation queries is a well-received technique. It is believed that sampling would bring more benefits to approximate queries as the volume of data increases. Our experience with production jobs in SCOPE, however, proves that this notion is naive. As a matter of fact, what we have observed (and reported in this paper) is that the simple-minded uniform sampling is ineffective, whose sample size keeps increasing with data volume, and at the end delivers no performance gains. The culprit is sparse data (with respective to data range) whose skewness in distribution gets severe as more and more data are included. An effective sampling scheme must understand the nature of the data, divide the data into regions, and sample them appropriately. The increased cost of sampling can be amortized over different queries that share them, and over time by using incremental updates. We have developed theoretical optimal as well as practical heuristic variants, and verified our techniques against real-world production jobs with a real implementation. The important lesson is that system optimizations are becoming increasingly more data-dependent.

8. ACKNOWLEDGEMENTS

We would like to thank Haixun Wang for his insightful comments on this work. We thank An Yan, Redmond Duan and Xudong Zheng for providing endless support during the investigation. We would also like to give our acknowledgements to Tao Yang and Xuzhan Sun for their help on the evaluation.

9. REFERENCES

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

[2] 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 *In Proc. of ACM EuroSys 2013*, 2013.

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

[4] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2), 2008.

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

[6] S. Chaudhuri, G. Das, and V. R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2), 2007.

[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *SIGMOD*, 2010.

[8] V. Ganti, M. L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, pages 176–187, 2000.

[9] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, pages 51–63. IEEE Computer Society Press, 1996.

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

[11] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58, 1963.

[12] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *PVLDB*, 5(10), 2012.

[13] S. L. Lohr. *Sampling: design and analysis*. Thomson Brooks/Cole, 2010.

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

[15] R.J.Serfling. Probability inequalities for the sum in sampling without replacement. *Institute of Mathematical Statistics*, 38, 1973.

[16] P. Rösch and W. Lehner. Sample synopses for approximate answering of group-by queries. In *EDBT*, 2009.

[17] L. Sidirourgos, M. Kersten, and P. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *In Proc. of the Intl Conf. on Innovative Data Systems Research (CIDR*, pages 296–301, 2011.

[18] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, Mar. 1985.

[19] Y. Wang, J. Luo, A. Song, J. Jin, and F. Dong. Improving online aggregation performance for skewed data distribution. In *DASFAA (1)*, volume 7238 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2012.

[20] S. Wu, S. Jiang, B. C. Ooi, and K.-L. Tan. Distributed online aggregation. *PVLDB*, 2(1):443–454, 2009.

[21] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*, 2010.

[22] J. Zhou, N. Bruno, M.-C. Wu, P.-Å. Larson, R. Chaiken, and D. Shakib. Scope: parallel databases meet mapreduce. *VLDB J.*, 21(5), 2012.

APPENDIX

A. SAMPLE MAINTENANCE ALGORITHM

The sliding window sample maintenance algorithm's pseudo code is shown in Algorithm 2. First, it computes a set of buckets $\{B_1, \ldots, B_q\}$, $q \leq k$ that need to be updated, together with their updated error bounds $\{\varepsilon'_1, \ldots, \varepsilon'_q\}$ (Line 2). It also computes the difference $\triangle_i = N_i^a - n_i^d$ of the updated buckets (Line 3). This information will be used for adjusting insertions in the next stage. The algorithm then checks Equation 13. If it is not satisfied, the algorithm moves to the next stage to compute n_i^a. In the second stage, the algorithm aims to find one or more buckets in $\mathcal{B}'$ with $\triangle_i \geq 0$ to help filling the gap of the error bound difference $\varepsilon_0 - \varepsilon'$ (Line 10 to 20). From Equation 13, the buckets with larger populations can contribute more to the global error bound. It means given the same number of insertions, inserting them to buckets with high populations is more effective in reducing the global error bound. To implement this heuristic, the algorithm reorders the buckets with $\triangle_i \geq 0$ by their populations, and tries to insert all incoming data into top-ranked buckets. If the gap $\varepsilon_0 - \varepsilon'$ can not be filled after all incoming data has been exhausted, the sketch is under sampling due to deletions, and must be reconstructed from scratch.

Algorithm 2: Maintenance algorithm for sliding windowed queries

Input: A list of bucket $\mathcal{B}$:$\{B_1, \ldots, B_k\}$
Output: Buckets $\mathcal{B}$ with updated error bounds.

1: **for** $i = 1$ to k **do**
2: $\quad \varepsilon_i = \varepsilon'_i \leftarrow$ Update the new error bound
3: $\quad \triangle_i = N_i^a - n_i^d$
4: **end for**
5: $\varepsilon' \leftarrow$ the global error bound with Equation 13
6: **if** $\varepsilon' \leq \varepsilon_0$ **then**
7: $\quad$ **return** $\mathcal{B}$
8: **end if**
9: Reorder the buckets whose $\triangle_i \geq 0$ according to its population $\{q$ buckets$\}$
10: **for** $i = 1$ to q **do**
11: $\quad \varepsilon'_i \leftarrow N\varepsilon_0 - \sum_{j=1, j\neq i}^{k} N_j \varepsilon_j$ {Compute ε'_i by Equation 6}
12: $\quad n'_i \leftarrow$ sample size with ε'_i by Hoeffding Equation 2
13: $\quad$ **if** $n'_i <= n_i + N_i^a$ **then**
14: $\quad\quad$ Update the error bound in B_i to ε'_i
15: $\quad\quad$ **return** $\mathcal{B}$
16: $\quad$ **else**
17: $\quad\quad n'_i = n_i + N_i^a$
18: $\quad\quad \varepsilon_i = \varepsilon'_i \leftarrow$ update the error bound
19: $\quad$ **end if**
20: **end for**
21: **return** NULL {Need to be reconstructed}