Effective Use of Block-Level Sampling in Statistics Estimation

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Gautam Das
Microsoft Research
gautamd@microsoft.com

Utkarsh Srivastava*
Stanford University
usriv@stanford.edu

ABSTRACT

Block-level sampling is far more efficient than true uniform-random sampling over a large database, but prone to significant errors if used to create database statistics. In this paper, we develop principled approaches to overcome this limitation of block-level sampling for histograms as well as distinct-value estimations. For histogram construction, we give a novel two-phase adaptive method in which the sample size required to reach a desired accuracy is decided based on a first phase sample. This method is significantly faster than previous iterative methods proposed for the same problem. For distinct-value estimation, we show that existing estimators designed for uniform-random samples may perform very poorly if used directly on block-level samples. We present a key technique that computes an appropriate subset of a block-level sample that is suitable for use with most existing estimators. This, to the best of our knowledge, is the first principled method for distinct-value estimation with block-level samples. We provide extensive experimental results validating our methods.

1. INTRODUCTION

Building database statistics by a full scan of large tables can be expensive. To address this problem, building approximate statistics using a random sample of the data is a natural alternative. There has been a lot of work on constructing statistics such as histograms and distinct values through sampling [1, 2, 7]. Most of this work deals with uniform-random sampling. However, true uniform-random sampling can be quite expensive . For example, suppose that there are 50 tuples per disk block and we are retrieving a 2% uniform-random sample. Then the expected number of tuples that will be chosen from each block is 1. This means that our uniform-random sample will touch almost every block of the table. Thus, in this case, taking a 2% uniform-random sample will be no faster than doing a full scan of the table.

Clearly, uniform-random sampling is impractical except for very small sample sizes. Therefore, most commercial relational database systems provide the ability to do *block-level sampling*, in which to sample a fraction of q tuples of a table, a fraction q of the disk-blocks of the table are chosen uniformly at random, and *all the tuples in these blocks* are returned in the sample. Thus, in contrast to uniform-random sampling, block-level sampling requires significantly fewer block accesses for the same sample size (blocks are typically quite large, e.g., 8K bytes).

The caveat is that a block-level sample is no longer a uniform sample of the table. The accuracy of statistics built over a block-level sample depends on the *layout* of the data on disk, i.e., the way tuples are grouped into blocks. In one extreme, a block-level sample may be just as good as a uniform-random sample, for example, when the layout is random, i.e., if there is no statistical dependence between the value of a tuple and the block in which it resides. However, in other cases, the values in a block may be fully correlated (e.g., if the table is clustered on the column on which the histogram is being built). In such cases, the statistics constructed from a block-level sample may be quite inaccurate as compared to those constructed from a uniform-random sample of the same size.

Given the high cost of uniform-random sampling, we contend that most previous work on statistics estimation through uniform-random sampling is of theoretical significance, unless robust and efficient extensions of those techniques can be devised to work with block-level samples. Surprisingly, despite the widespread use of block-level sampling in relational products, there has been limited progress in database research in analyzing impact of block-level sampling on statistics estimation. An example of past work is [2]; however it only addresses the problem of histogram construction from block-level samples, and the suggested scheme carries a significant performance penalty.

In this paper, we take a comprehensive look at the significant impact of block-level sampling on statistics estimation. To effectively build statistical estimators with block-level sampling, the challenge is to leverage the sample as efficiently as possible, and still be robust in the presence of any type of correlations that may be present in the sample. Specifically, we provide a foundation for developing principled approaches that leverage block-level samples for histogram construction as well as distinct-value estimation.

For histogram construction, the main challenge is in determining the *required sample size* to construct a histogram with a desired accuracy: if the layout is fairly random then a small sample will suffice, whereas if the layout is highly correlated, a much larger sample is needed. We propose a 2-phase sampling algorithm that is significantly more efficient (200% or more) than what was proposed in [2]. In the first phase, our algorithm uses an initial block-level sample to determine "how much more to sample" by using cross-validation techniques. This phase is optimized so that the cross-validation step can piggyback on a standard sort-based algorithm for building histograms. In the second and final phase, the al-

*This work was done while the author was visiting Microsoft Research

SIGMOD 2004 June 13-18, 2004, Paris, France.

gorithm uses block-level sampling to gather the remaining sample, and build the final histogram. This is in sharp contrast to the algorithm in [2] that blindly doubles the sample size iteratively until the desired accuracy is reached— thus increasing sample size significantly, and paying significant overheads at each iteration. We back up our rationale for 2-phase histogram construction with a formal analytical model, and demonstrate its overwhelming superiority experimentally.

Distinct-value estimation is fundamentally different from histogram construction. To the best of our knowledge, despite a very large body of work on many distinct-value estimators for uniform-random sampling [1, 7, 11, 15], no past work has analyzed the impact of block-level sampling on such estimators. We formally show that using such estimators directly on the entire block-level sample may yield significantly worse estimates compared to those obtained by using them on an "appropriate subset" of the block-level sample. Our experiments confirm that our procedure for selecting such an appropriate subset does indeed result in distinct-value estimates that are almost as accurate as estimates obtained from uniform-random samples of similar size, and often vastly better than the estimates obtained by the naïve approach of applying the estimator on the entire block-level sample.

Finally, our study led to the identification of novel measures that quantify the "degree of badness" of the layout for block-level sampling for statistics estimation. Interestingly, these measures are found to be different for histograms and distinct-values, thus emphasizing the fundamental differences between the two problems.

The rest of the paper is organized as follows. In Section 2, we survey related work. In Section 3, we investigate the problem of histogram construction, and in Section 4, the problem of distinct-value estimation, with block-level samples. We have prototyped our algorithms using Microsoft SQL Server. We present the experimental results in Section 5, and conclude in Section 6.

2. RELATED WORK

Random sampling has been used for solving many database problems. In statistics literature, the concept of *cluster sampling* is similar to block-level sampling being considered here [3]. A large body of work addresses the problem of estimating query-result sizes by sampling [8, 9, 10, 12]. The idea of using cluster sampling to improve the utilization of the sampled data, was first proposed for this problem by Hou et. al. [10]. However, they focus on developing consistent and unbiased estimators for COUNT queries, and the approach is not error driven. For distinct-value estimation with block-level samples, they simply use the Goodman's estimator [6] directly on the block-level sample, recognizing that such an approach can lead to a significant bias. The use of *two-phase*, or *double* sampling was first proposed by Hou et. al. [9], also in the context of COUNT query evaluation. However, their work considers uniform-random samples instead of block-level samples, and does not directly apply to histogram construction.

The use of random sampling for histogram construction was first proposed by Piatetsky-Shapiro et. al. [13]. In this context, the problem of deciding how much to sample for a desired error, has been addressed in [2, 5]. However, these derivations assume uniform random sampling. Only Chaudhuri et. al. [2] consider the problem of histogram construction through block-level sampling. They propose an iterative cross-validation based approach to arrive at the correct sample size for the desired error. However, in contrast to our two-phase approach, their approach often goes through a large number of iterations to arrive at the correct sample size, consequently incurring much higher overhead. Also, their approach frequently samples more than required for the desired error.

The problem of distinct value-estimation through uniform random sampling has received considerable attention [1, 6, 7, 11, 15]. The Goodman's estimator [6] is the unique unbiased distinct-value estimator for uniform random samples. However, it is unusable in practice [7] due to its extremely high variance. The hardness of distinct-value estimation has been established by Charikar et. al. in [1]. Most estimators that work well in practice do not give any analytical error guarantees. For the large sampling fractions that distinct-value estimators typically require, uniform-random sampling is impractical. Haas et. al. note in [7] that their estimators are useful only when the relation is laid out randomly on disk (so that a block-level random sample is as good as a uniform-random sample). However, distinct value estimation through block-level sampling has remained unaddressed. To the best of our knowledge, our work is the first to address this problem in a principled manner.

3. HISTOGRAM CONSTRUCTION

3.1 Preliminaries

Almost all query-optimization methods rely on the availability of statistics on database columns to choose efficient query plans. Histograms have traditionally been the most popular means of storing these statistics compactly, and yet with reasonable accuracy. Any type of histogram can essentially be viewed as approximation of the underlying data distribution, and is a partitioning of the domain into disjoint *buckets* and storing the counts of the number of tuples belonging to each bucket. These counts are often augmented with *density* information, i.e., the average number of duplicates for each distinct value. To estimate density, a knowledge of the number of *distinct values* in the relevant column is required. Bucket counts help in cardinality estimation of range queries while density information helps for equality queries.

Histogram algorithms differ primarily in how the bucket separators are selected to reduce the error in approximating the underlying data distribution. For example, an *equi-width* bucketing algorithm forms buckets with equal ranges, an *equi-depth* bucketing algorithm forms buckets with equal number of tuples, a *maxdiff* bucketing algorithm places separators where tuple frequencies on either side differ the most, while the optimal *v-opt* algorithm places separators such that this error is minimized [14].

3.1.1 Error-Metrics

We distinguish between two types of errors of histograms. The first type of error measures how accurately a histogram captures the underlying data distribution. The second type of error arises when the histogram is constructed through sampling. This error measures to what degree a histogram constructed over a sample, approximates a histogram constructed by a full scan of the data (i.e., a *perfect* histogram). In this paper, we are concerned with the second type of error.

Various metrics have been proposed for the second type of error. We first develop some notation. Consider a table with n tuples, containing an attribute X over a totally ordered domain $\mathcal{D}$. An approximate k-bucket histogram over the table is constructed through sampling as follows. Suppose a sample of r tuples is drawn. A bucketing algorithm uses the sample to decide a sequence of separators $s_1, s_2, \ldots, s_{k-1} \in \mathcal{D}$. These separators partition $\mathcal{D}$ into k buckets $B_1, B_2, \ldots, B_k$ where $B_i = \{v \in \mathcal{D} | s_{i-1} < v \leq s_i\}$ (We take $s_0 = -\infty$ and $s_k = \infty$). Let $\tilde{n}_i$ be the size of (i.e., number of tuples contained in) B_i in the sample, and n_i be the size of B_i in the table. The histogram estimates n_i as $\hat{n}_i = \frac{n}{r} \cdot \tilde{n}_i$. The histogram is *perfect* if $n_i = \hat{n}_i$ for $i = 1, 2, \ldots, k$.

The *variance-error* metric [5] measures the mean squared error

across all buckets, normalized with respect to the mean bucket size:

$$\Delta_{var} = \frac{k}{n}\sqrt{\frac{1}{k}\sum_{i=1}^{k}(\hat{n}_i - n_i)^2} \quad (1)$$

For the special case of equi-depth histograms, the problem of deriving the uniform-random sample size required to reach a given variance-error with high probability has been considered in [5].

The *max-error* metric [2] measures the maximum error across all buckets:

$$\Delta_{max} = \max_i \left\{ \frac{|\hat{n}_i - n_i|}{(n/k)} \right\} \quad (2)$$

For equi-depth histograms, the uniform-random sample size needed to reach a desired max-error with high probability is derived in [2].

Although the methods developed in our paper can work for both kinds of metrics, in practice we observed that the max-error metric was overly conservative: a single bad bucket unduly penalizes a histogram whose accuracy is otherwise tolerable in practice. Conversely, an unreasonably large sample size is often required to achieve a desired error bound. This was especially true when the layout was "bad" for block-level sampling (see Section 3.1.2). Due to these difficulties with the max-error metric, in the rest of this paper we chose to describe our results only for the variance-error metric.

3.1.2 Problem Formulation

The layout of a database table (i.e., the way the tuples are grouped into blocks) can significantly affect the error in a histogram constructed over a block-level sample. This point was recognized in [2], and is illustrated by the following two extreme cases:

- **Random Layout**: For a table in which the tuples are grouped randomly into blocks, a block-level sample is equivalent to a uniform-random sample. In this case, a histogram built over a block-level sample will have the same error as a histogram built over a uniform-random sample of the same size.
- **Clustered Layout**: For a table in which all tuples in a block have the same value in the relevant attribute, sampling a full block is equivalent to sampling a single tuple from the table (since the contents of the full block can be determined given one tuple of the block). In this case, a histogram built over a block-level sample will have a higher error as compared to one built over a uniform-random sample of the same size.

In practice, most real layouts fall somewhere in between. For example, suppose a relation was clustered on the relevant attribute, but at some point in time the clustered index was dropped. Now suppose inserts to the relation continue to happen. This results in the table becoming "partially clustered" on this attribute. As another example, consider a table which has columns Age and $Salary$, and is clustered on the Age attribute. Since an older age usually (but not always) implies a higher salary, the table shall be "almost clustered" on $Salary$ too.

Suppose, we have to construct a histogram with a desired error bound. The above arguments show that the block-level sample size required to reach the desired error depends significantly on the layout of the table. In this section, we consider the problem of constructing a histogram with the desired error bound through block-level sampling, by adaptively determining the required block-level sample size according to the layout.

The rest of this section is organized as follows. In the next subsection we briefly describe an iterative cross-validation based approach (previously developed in [2]) for this problem, and discuss its shortcomings. In Section 3.3, we provide the formal analysis which motivates our solution to the problem. Our proposed algorithm 2PHASE is given in Section 3.4.

3.2 Cross-Validation Based Iterative Approach

The idea behind cross-validation is the following. First, a block-level sample S_1 of size r is obtained, and a histogram $\mathcal{H}$ is constructed on it. Then another block-level sample S_2 of the same size is drawn. Let $\tilde{n}_i$ (resp. $\tilde{m}_i$) be the size of the i^{th} bucket of $\mathcal{H}$ in S_1 (resp. S_2). Then, the cross-validation error according to the variance error-metric is given by:

$$\Delta_{var}^{CV} = \frac{k}{r}\sqrt{\frac{1}{k}\sum_{i=1}^{k}(\tilde{n}_i - \tilde{m}_i)^2} \quad (3)$$

Intuitively, the cross-validation error measures the similarity of the two samples in terms of the value distribution. Cross-validation error is typically higher than the actual variance error [2]: it is unlikely for two independent samples to resemble each other in distribution, but not to resemble the original table. Based on this fact, a straightforward algorithm has been proposed in [2] to arrive at the required block-level sample size for a desired error. Let r_{unf} (resp. r_{blk}) be the uniform-random sample size (resp. block-level sample size) required to reach the desired error. The algorithm starts with an initial block-level sample of size r_{unf}. The sample size is *repeatedly doubled* and cross-validation performed, until the cross-validation error reaches the desired error target. Henceforth, we shall refer to this algorithm as DOUBLE. The major limitation of DOUBLE is that it always increases the sample size by a factor of two. This blind step factor hurts in both the following cases:

- Each iteration of the algorithm incurs considerable fixed overheads of drawing a random block-level sample, sorting the incremental sample, constructing a histogram, and performing the cross-validation test. For significantly clustered data where r_{blk} is much larger than r_{unf}, the number of iterations becomes a critical factor in the performance.
- If at some stage in the iterative process, the sample size is close to r_{blk}, the algorithm is oblivious of this, and samples more than required. In fact in the worst case, the total sample drawn maybe four times r_{blk}, because an additional sample of the same size is required for the final cross-validation.

To remedy these limitations, the challenge is to develop an approach which (a) goes through a much smaller number of iterations (ideally one or two) so that the effect of the overhead per iteration is minimized, and (b) does not overshoot the required sample size by much. Clearly, these requirements can be met only if our algorithm has a knowledge of how the cross-validation error decreases with sample size. We formally develop such a relationship in the following subsection, which provides the motivation for our eventual algorithm, 2PHASE.

3.3 Motivating Formal Analysis

In this subsection we formally study the relationship between cross-validation error and sample size. To keep the problem analyzable, we adopt the following simplified model: we assume that the histogram construction algorithm is such that the histograms produced over any two different samples have the *same bucket separators*, and differ only in the estimated counts of the corresponding buckets. For example, an equi-width histogram satisfies this assumption. This assumption is merely to motivate our analysis of the proposed algorithm. However, our algorithm itself can can work

with any histogram construction algorithm, and does not actually fix bucket boundaries. Indeed, our experimental results (Section 5) demonstrate the effectiveness of our approach for both equi-depth histograms and maxdiff histograms, neither of which satisfies the above assumption of same bucket separators. Of course, it is an interesting open problem whether these histograms can be formally analyzed for sampling errors without making the above simplifying assumption.

Recall the notation introduced in Section 3.1.1. Let there be n tuples and N blocks in the table with b tuples per block ($N = n/b$). Given a histogram $\mathcal{H}$ with k buckets, consider the distribution of the tuples of bucket B_i among the blocks. Let a fraction a_{ij} of the tuples in the j^{th} block belong to bucket B_i ($n_i = b \cdot \sum_{j=1}^{N} a_{ij}$). Let σ_i^2 denote the variance of the numbers $\{a_{ij} | j = 1, 2, \ldots, N\}$. Intuitively, σ_i^2 measures how evenly the tuples of bucket B_i are distributed among the blocks. If they are fairly evenly distributed, σ_i^2 will be small. On the other hand, if they are concentrated in relatively few blocks, σ_i^2 will be large.

Let S_1 and S_2 be two independent block-level samples of r tuples each. We assume blocks are sampled with replacement. For large tables, this closely approximates the case of sampling without replacement. Suppose we construct a histogram $\mathcal{H}$ over S_1, and cross-validate it against S_2. Let Δ_{var}^{CV} be the cross-validation error obtained.

THEOREM 1. $E[(\Delta_{var}^{CV})^2] = \frac{2kb}{r} \sum_i \sigma_i^2$

PROOF. Let $\tilde{n}_i$ (resp. $\tilde{m}_i$) be the size of B_i in S_1 (resp. S_2). For fixed bucket separators, both $\tilde{n}_i$ and $\tilde{m}_i$ have the same distribution. We first find the mean and variance of these variables. The mean is independent of the layout, and is given by

$$\mu_{\tilde{n}_i} = \mu_{\tilde{m}_i} = \frac{r}{n} \cdot n_i$$

The expression for the variance is more involved and depends on the layout. A block-level sample of r tuples consists of r/b blocks chosen uniformly at random. If block j is included in the block-level sample, it contributes ba_{ij} tuples to the size of B_i. Thus, $\tilde{n}_i$ (or $\tilde{m}_i$) is equal to b times the sum of r/b independent draws with replacement from the a_{ij}'s. Hence, by the standard sampling theorem [3],

$$\sigma_{\tilde{n}_i}^2 = \sigma_{\tilde{m}_i}^2 = \frac{r}{b} \cdot b^2 \cdot \sigma_i^2 = rb\sigma_i^2$$

By Equation 3 for the cross-validation error:

$$\begin{aligned} E[(\Delta_{var}^{CV})^2] &= \frac{k}{r^2} \sum_{i=1}^{k} E[(\tilde{n}_i - \tilde{m}_i)^2] \\ &= \frac{k}{r^2} \sum_{i=1}^{k} E[(\tilde{n}_i - \mu_{\tilde{n}_i})^2] + E[(\tilde{m}_i - \mu_{\tilde{m}_i})^2] \\ &= \frac{k}{r^2} \sum_{i=1}^{k} \sigma_{\tilde{n}_i}^2 + \sigma_{\tilde{m}_i}^2 \qquad (4) \\ &= \frac{2kb}{r} \sum_{i=1}^{k} \sigma_i^2 \end{aligned}$$

□

There are three key conclusions from this analysis:

1. The expected squared cross-validation error is inversely proportional to the sample size. This forms the basis of a more intelligent step factor than the blind factor of two in the iterative approach of [2].
2. In Equation 4, the first term inside the summation represents the actual variance-error. Since both terms are equal in expectation, the cross-validation error can be expected to be about $\sqrt{2}$ times the actual variance-error. Thus it is sufficient to stop sampling when the cross-validation error has reached the desired error target.
3. The quantity $\sum_{i=1}^{k} \sigma_i^2$ represents a quantitative measure of the "badness" of a layout for constructing the histogram $\mathcal{H}$. If this quantity is large, the cross-validation error (and also the actual variance-error) is large, and we need a bigger block-level sample for the same accuracy. Besides the layout, this measure also naturally depends on the bucket separators of $\mathcal{H}$. Henceforth we refer to this quantity as *Hist_Badness*.

We next describe our 2PHASE algorithm for histogram construction, which is motivated by the above theoretical analysis.

3.4 The 2PHASE Algorithm

Suppose we wish to construct a histogram with a desired error threshold. For simplicity, we assume that the threshold is specified in terms of the desired cross-validation error Δ^{req} (since the actual error is typically less). Theorem 1 gives an expression for the expected squared cross-validation error, i.e., it is proportional to *Hist_Badness* and inversely proportional to the block-level sample size. Since in general we do not know *Hist_Badness* (such information about the layout is almost never directly available), we propose a 2-phase approach: draw an initial block-level sample in the first phase and use it to try and estimate *Hist_Badness* (and consequently the required block-level sample size), then draw the remaining block-level sample and construct the final histogram in the second phase. The performance of this overall approach critically depends on how accurate the first phase is in determining the required sample size. An accurate first phase would ensure that this approach is much superior to the cross-validation approach of [2] because (a) there are far fewer iterations and therefore significantly fewer overheads, (b) the chance of overshooting the required sample size is reduced, and (c) there is no final cross-validation step to check whether the desired accuracy has been reached.

A straightforward implementation of the first phase might be as follows. We pick an initial block-level sample of size $2r_{unf}$ (where r_{unf} is the theoretical sample size that achieves an error of Δ^{req} assuming uniform-random sampling). We divide this initial sample into two halves, build a histogram on one half and cross-validate this histogram using the other half. Suppose the observed cross-validation error is Δ^{obs}. If $\Delta^{obs} \leq \Delta^{req}$ we are done, otherwise the required block-level sample size r_{blk} can be derived from Theorem 1 to be $(\frac{\Delta^{obs}}{\Delta^{req}})^2 \cdot r_{unf}$. However, this approach is not very robust. Since Theorem 1 holds only for *expected* squared cross-validation error, using a single estimate of the cross-validation error to predict r_{blk} may be very unreliable. Our prediction of r_{blk} should ideally be based on the mean of a number of trials.

To overcome this shortcoming, we propose our 2PHASE algorithm, in which the first phase performs many cross-validation trials for estimating r_{blk} accurately. However, the interesting aspect of our proposal is that this robustness *comes with almost no performance penalty*. A novel scheme is employed in which multiple cross-validations are piggybacked on sorting, so that the resulting time complexity is comparable to that of a single sorting step. Since most histogram construction algorithms require sorting anyway[1],

[1]Equi-depth histograms are exceptions because they can be constructed by finding quantiles. However, in practice equi-depth histograms are often implemented by sorting [2].

Algorithm 2PHASE

Input:

Δ^{req} : Desired maximum cross-validation error in histogram
r_1 : Input parameter for setting initial sample size
l_{max} : Number of points needed to do curve-fitting

Phase I:

1. $A[1 \ldots 2r_1]$ = block-level sample of $2r_1$ tuples
2. sortAndValidate($A[1 \ldots 2r_1], 0$)
3. r_{blk} = getRequiredSampleSize()

Phase II:

4. $A[2r_1 + 1 \ldots r_{blk}]$ = block-level sample of $r_{blk} - 2r_1$ tuples
5. sort($A[2r_1 + 1 \ldots r_{blk}]$)
6. merge($A[1 \ldots 2r_1], A[2r_1 + 1 \ldots r_{blk}]$)
7. createHistogram($A[1 \ldots r_{blk}]$)

sortAndValidate($A[1 \ldots r], l$)
1. if ($l = l_{max}$)
2. sort($A[1 \ldots r]$)
3. else
4. $m = \lfloor r/2 \rfloor$
5. sortAndValidate($A[1 \ldots m], l + 1$)
6. sortAndValidate($A[m + 1 \ldots r], l + 1$)
7. lh = createHistogram($A[1 \ldots m]$)
8. rh = createHistogram($A[m + 1 \ldots r]$)
9. $sqErr[l]$ += getSquaredError($lh, A[m + 1 \ldots r]$)
10. $sqErr[l]$ += getSquaredError($rh, A[1 \ldots m]$)
11. merge($A[1 \ldots m], A[m + 1 \ldots r]$)

getRequiredSampleSize()
1. if ($sqErr[0]/2 \leq (\Delta^{req})^2$)
2. return $2r_1$
3. else
4. Fit a curve of the form $y = c/x$ through the points $(r_1/2^i, sqErr[i]/2^{i+1})$ for $i = 0, 1, \ldots, l_{max} - 1$
5. return $\frac{c}{(\Delta^{req})^2}$

Figure 1: 2-Phase approach to sampling for histogram construction

this sharing of cross-validation and sorting leads to a very robust yet efficient approach.

The pseudo-code for 2PHASE is shown in Figure 1. We assume *merge*, *sort*, *createHistogram* and *getSquaredError* are externally supplied methods. The first two have their standard functionality. The function *createHistogram* can be any histogram construction algorithm such as the equi-depth algorithm, or the maxdiff algorithm [14]. The function *getSquaredError* cross-validates the given histogram against the given sample, and returns the squared cross-validation error $(\Delta_{var}^{CV})^2$, according to Equation 3.

In Phase I, the algorithm picks an initial block-level sample of size $2r_1$ where r_1 is an input parameter. This parameter can be set as r_{unf}, however in practice we found that a setting that is 2 to 3 times larger yields much more robust results. Then, cross-validation is performed on different size subparts of the initial sample, where the task of cross-validation is combined with that of sorting. This piggybacking idea is illustrated in Figure 2, and is implemented by the *sortAndValidate* procedure in Figure 1. We use an in-memory merge-sort for sorting the sample (the sample sizes used in the first phase easily fit in memory). To sort and cross-validate a sample of size r, it is divided into two halves. Each of these are recursively sorted and cross-validated. Then, histograms are built

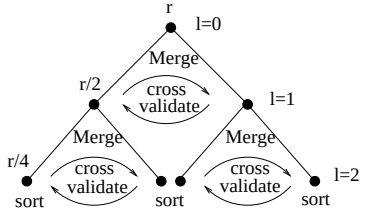

Figure 2: Combining cross-validation with sorting for $l_{max} = 2$

on the left and right halves. Each histogram is tested against the other half, and two estimates of $(\Delta_{var}^{CV})^2$ for a sample of size $r/2$ are obtained. Note that the recursive cross-validation of the two halves will give several $(\Delta_{var}^{CV})^2$ estimates for each sample size $r/4, r/8 \ldots$ etc. Effectively, we are reusing subparts of the sample to get several different cross-validation error estimates. We note that the standard statistical technique of *bootstrap* is also based upon reusing different subparts of a sample [4], and it would be interesting to explore its connections with our technique. However, the approach of piggybacking on merge-sort is very specific to our technique, and is motivated by efficiency considerations.

Although quick-sort is typically the fastest in-memory sort (and is the method of choice in traditional in-memory histogram construction algorithms), merge-sort is not much slower. Moreover, it allows us to combine cross-validation with sorting. The merge-sort is parameterized to not form its entire recursion tree, but to truncate after the number of levels has increased to a threshold (l_{max}). This reduces the overall overhead of cross-validation. Also, at lower sample sizes, error estimates lose statistical significance. Usually a small number such as $l_{max} = 3$ suffices for our purposes. At the leaves of the recursion tree, we perform quick-sort rather than continuing with merge-sort.

Once this sorting phase is over, we have several $(\Delta_{var}^{CV})^2$ estimates corresponding to each sample size $r_1, r_1/2, \ldots, r_1/2^{l_{max}-1}$. We compute the mean of these estimates for each of these sample sizes. We then find the best fitting curve of the form $\Delta^2 = c/r$ (justified by Theorem 1) to fit our observed points, where c is a constant, and Δ^2 is the average squared cross-validation error observed for a sample of size r. This curve fitting is done using the standard method of least-squares. The best-fit curve yields a value of c which is used to predict r_{blk} by putting $\Delta = \Delta^{req}$. This is done in the procedure *getRequiredSampleSize*.

Finally, once we have an estimate for r_{blk}, we enter Phase II. The additional sample required (of size $r_{blk} - 2r_1$) is obtained and sorted. It is merged with the (already sorted) first-stage sample, a histogram is built on the total sample, and returned.

In summary, the 2PHASE algorithm is significantly more efficient than DOUBLE, mainly because it uses a more intelligent step factor that enables termination after only two phases. Note that 2PHASE seeks to reach the cross-validation error target in the expected sense, thus there is a theoretical possibility that the error target may not be reached after the second phase. One way to avoid this problem would be to develop a high probability bound on the cross-validation error (rather than just an expected error bound as in Theorem 1), and modify the algorithm accordingly so that it reaches the error target with high probability. Another alternative would be to extend 2PHASE to a potentially multi-phase approach, where the step size is decided as in 2PHASE, but the termination criterion is based on a final cross-validation step as in DOUBLE. Although this will reduce the number of iterations as compared to DOUBLE, it will still not solve the problem of oversampling due to the final cross-validation step. However, neither of these extensions seem to be necessary since 2PHASE in its present form almost al-

ways reaches the cross-validation error target in practice. Even in the few cases in which it fails, the *actual* variance-error (which is typically substantially smaller than the cross-validation error) is always well below the error target.

4. DISTINCT VALUE ESTIMATION

4.1 Problem Formulation

The number of *distinct-values* is a popular statistic commonly maintained by database systems. Distinct-value estimates often appear as part of histograms, because in addition to tuple counts in buckets, histograms also maintain a count of the number of distinct values in each bucket. This gives a density measure for each bucket, which is defined as the average number of duplicates per distinct value. The bucket density is returned as the estimated cardinality of any query with a selection predicate of the form $X = a$, where a is any value in the range of the bucket, and X is the attribute over which the histogram has been built. Thus, any implementation of histogram construction through sampling must also solve the problem of estimating the number of distinct values in each bucket.

There has been a large body of work on distinct-value estimation using uniform-random sampling [1, 6, 7, 11, 15]. Here we address the *different* problem of distinct-value estimation through block-level sampling. To the best of our knowledge, this problem has not been addressed in a principled manner in previous work. We shall only consider the problem of estimating the number of distinct values on the entire column X through block-level sampling. The most straightforward way to extend it to histogram buckets is to use the distinct value estimators on subparts of the sample corresponding to each bucket.

We clarify that this problem is different in flavor compared to the one we addressed for histogram construction. Here we focus on developing the best distinct-value estimator to use with block-level samples. The problem of deciding how much to sample to reach a desired accuracy (which we had addressed for histograms), remains open for future work. This seems to crucially depend on analytical error guarantees, which are unavailable for most distinct-value estimators even with uniform-random sampling [1, 7].

Let D be the number of distinct values in the column, and let $\hat{D}$ be the estimate returned by an estimator. We distinguish between the *bias* and *error* of the estimator:

$$\begin{aligned} \text{Bias} &= |E[\hat{D}] - D| \\ \text{Error} &= \max\{\hat{D}/D, D/\hat{D}\} \end{aligned}$$

Our definition of error is according to the *ratio-error* metric defined in [1]. A perfect estimator shall have error $= 1$. Notice that it is possible for an estimator to be unbiased (i.e. $E[\hat{D}] = D$), but still have high expected error.

Most prior work has been to develop estimators with small bias for uniform-random sampling. Getting a bound on the error is considerably harder [1, 7]. In fact, there are no known estimators that guarantee error bounds even for uniform-random sampling [2]. Ideally, we would like to leverage existing estimators which have been designed for uniform-random samples and make them work for block-level samples. Moreover, we seek to use these estimators with block-level samples in such a way, that the bias and error are not much larger than when these estimators are used with uniform-random samples of the same size.

The rest of this section is organized as follows. In the next subsection, we show that if existing distinct-value estimators are used naïvely with block-level samples, highly inaccurate estimates may be produced. Then, in Section 4.3, we develop an exceedingly simple yet novel technique called COLLAPSE. Using formal arguments, we show that COLLAPSE allows us to use a large class of existing estimators on block-level samples instead of uniform-random samples such that the bias remains small. Finally, in Section 4.4, we study the performance of COLLAPSE in terms of the ratio-error metric. As with histograms, we identify a novel measure that quantifies the "degree of badness" of the layout for block-level sampling for distinct-value estimation. Interestingly, this measure is found to be different from the corresponding measure for histograms, thus emphasizing the fundamental differences between the two problems.

[2]The formal result for the GEE estimator in [1] is a proof of the bias being bounded, not error.

4.2 Failure of Naive Approach

Consider the following naive approach (called TAKEALL) for distinct-value estimation with block-level sampling:

TAKEALL: *Take a block-level sample S_{blk} with sampling fraction q. Use S_{blk} with an existing estimator as if it were a uniform-random sample with sampling fraction q.*

We show that many existing estimators may return very poor estimates if used with TAKEALL. Our arguments apply to most estimators which have been experimentally evaluated, and found to perform well on uniform-random samples, e.g., the HYBSKEW estimator [7], the smoothed jackknife estimator [7, 11], the Shlosser estimator [15], the GEE estimator [1], and the AE estimator [1].

Let d be the number of distinct values in the sample. Let there be f_i distinct values which occur exactly i times in the sample. All the estimators mentioned above have the common form $\hat{D} = d + K \cdot f_1$, where K is a constant chosen adaptively according to the sample (or fixed according to the sampling fraction as in GEE). The rationale behind this form of the estimators is as follows. Intuitively, f_1 represents the values which are "rare" in the entire table (have low multiplicity), while the higher frequency elements in the sample represent the values which are "abundant" in the table (have high multiplicity). A uniform-random sample is expected to have missed only the rare values, and none of the abundant values. Hence we need to scale-up only the rare values to get an estimate of the total number of distinct values.

However, this reasoning does not apply when these estimators are used with TAKEALL. Specifically, consider a table in which the multiplicity of every distinct value is at least 2. Further, consider a layout of this table such that for each distinct value, its multiplicity in any block is either 0 or at least 2. For this layout, in any block-level sample (of any size), $f_1 = 0$. Thus, in this case, all the above estimators will return $\hat{D} = d$. Effectively, no scaling is applied, and hence the resulting estimate may be highly inaccurate.

More generally, the reason why these estimators fail when used with TAKEALL, is as follows. When a particular occurrence of a value is included in a block-level sample, any more occurrences of the value in that block are also picked up— but by virtue of being present in that block, and not because that value is frequent. Thus, multiplicity across blocks is a good indicator of abundance, but multiplicity within a block is a misleading indicator of abundance.

4.3 Proposed Solution: COLLAPSE

In this section, we develop a very simple yet novel approach called COLLAPSE which enables us to use existing estimators on block-level samples instead of uniform-random samples.

The reasons for the failure of TAKEALL given in the previous subsection, suggest that to make the existing estimators work, a value should be considered abundant only if it occurs in multiple blocks in the sample, while multiple occurrences within a block

Algorithm COLLAPSE

Input: q: Block-level sampling fraction

1. *Sampling Step*: Take a block-level sample S_{blk} with sampling fraction q.
2. *Collapse Step*: In S_{blk}, collapse all multiple occurrences of a value within a block into one occurrence. Call the resulting sample S_{coll}.
3. *Estimation Step*: Use S_{coll} with an existing estimator as if it were a uniform-random sample with sampling fraction q.

Figure 3: Distinct-value estimation with block-level samples

should be considered as only a single occurrence. We refer to this as the *collapsing* of multiplicities within a block.

In fact, we can show that such a collapsing step is *necessary*, by the following adversarial model: If our estimator depends on the multiplicities of values within blocks, an adversary might adjust the multiplicities within the sampled block so as to hurt our estimate the most, while still not changing the number of distinct values in the table. For example, if our estimate scales only f_1 (as most existing estimators), the adversary can give a multiplicity of at least 2 to as many of the values in the block as possible. Thus, our estimator should be independent of the multiplicities of the values within blocks.

This leads us to develop a very simple approach called COLLAPSE shown in Figure 3. Essentially, multiplicities within blocks of a block-level sample are first collapsed, and then existing estimators are directly run on the collapsed sample, i.e., the collapsed sample is simply treated as if it were a uniform-random sample with the same sampling fraction as the block-level sample.

We now provide a formal justification of COLLAPSE. Let T be the table on which we are estimating the number of distinct values. Let v_j denote the j^{th} distinct value. Let n_j be the tuple-level multiplicity of v_j, i.e., the number of times it occurs in T, and N_j be the block-level multiplicity of v_j, i.e., the number of blocks of T in which it occurs. Let S_{blk} be a block-level sample from T with sampling fraction q, and S_{coll} be the sample obtained after applying the collapse step to S_{blk}. Let T_{coll} be an imaginary table obtained from T by collapsing multiple occurrences of values within every block into a single occurrence. Let S_{unf} be a uniform-random sample from T_{coll} with the same sampling fraction q. Notice that T_{coll} may have variable-sized blocks, but this does not affect our analysis. As before, let f_i denote the number of distinct values which occur exactly i times in a sample.

LEMMA 1. *For the Bernoulli sampling model, $E[f_i$ in $S_{coll}] = E[f_i$ in $S_{unf}]$ for all i.*

PROOF. In the Bernoulli sampling model, for picking a sample with sampling fraction q, each item is included with probability q independent of other items. This closely approximates uniform-random sampling for large table sizes.

A particular distinct value v_j contributes to f_i in S_{coll} iff exactly i blocks in which it occurs are chosen in S_{blk}. Since v_j occurs in N_j blocks, it contributes to $f_i (i \leq N_j)$ in S_{coll} with probability $\binom{N_j}{i} q^i (1-q)^{N_j - i}$. Thus,

$$E[f_i \text{ in } S_{coll}] = \sum_{j | N_j \geq i} \binom{N_j}{i} q^i (1-q)^{N_j - i}$$

Now, in T_{coll}, the tuple-level multiplicity of v_j is N_j. Thus, v_j contributes to f_i in S_{unf} iff exactly i occurrences out of its N_j occurrences are chosen in S_{unf}. Since the sampling fraction is q, the probability that v_j contributes to f_i in S_{unf} is the same as in the above. Hence the expected value of f_i in S_{unf} is the same as in S_{coll}. □

Now consider any distinct-value estimator $\mathcal{E}$ of the form $\hat{D} = \sum_{i=1}^{r} a_i f_i$ (where a_i's are constants depending on the sampling fraction). We can show that for use with estimator $\mathcal{E}$, S_{coll} is as good as S_{unf} (in terms of bias). Let $\mathcal{B}(T_{coll}, q)$ be the bias of $\mathcal{E}$ when applied to uniform-random samples from T_{coll} with sampling fraction q. Let $\mathcal{B}_{coll}(T, q)$ be the bias of $\mathcal{E}$ when applied to block-level samples from T with sampling fraction q, and which have been processed according to the collapse step.

THEOREM 2. $\mathcal{B}(T_{coll}, q) = \mathcal{B}_{coll}(T, q)$.

PROOF. First note that T_{coll} and T have the same number of distinct values. Further, by Lemma 1, $E[f_i$ in $S_{coll}] = E[f_i$ in $S_{unf}]$. $\mathcal{E}$ is just a linear combination of f_i's, and the coefficients depend only on the sampling fraction which is the same for S_{coll} and S_{unf}. Thus, by linearity of expectations, the result follows. □

The above theorem enables us to leverage much of previous work on distinct-value estimation with uniform-random samples. Most of this work [1, 7] tries to develop estimators with small bias on uniform-random samples. By Theorem 2, we reduce the problem of distinct-value estimation using block-level samples to that of distinct-value estimation using uniform-random samples of a modified (i.e., collapsed) table. For example, GEE [1] is an estimator which has been shown to have a bias of at most $O(\sqrt{1/q})$ on uniform-random samples with sampling fraction q. Moreover, GEE is of the form as required by Theorem 2 ($\hat{D}_{GEE} = \sum_{i=2}^{r} f_i + \frac{1}{\sqrt{q}} f_1$). Thus, if we use GEE with COLLAPSE, our estimate also will be biased by at most $O(\sqrt{1/q})$ for a block-level sampling fraction of q. Other estimators like HYBSKEW and AE do not exactly satisfy the conditions of Theorem 2 since the a_i's themselves depend on the f_i's. However, these estimators are heuristic anyway. Hence we experimentally compare the performance of COLLAPSE with these estimators, against using these estimators on uniform-random samples. The experimental results given in Section 5 demonstrate the superiority of COLLAPSE against TAKEALL with these estimators.

4.4 Studying Error for COLLAPSE

In this subsection we discuss the impact of COLLAPSE on the ratio-error of estimators. However, unlike bias, formal analysis of the ratio-error is extremely difficult even for uniform-random sampling [1, 7]. Consequently, much of the discussion in this subsection is limited to qualitative arguments. The only quantitative result we give is a lower bound on the error of any estimator with block-level sampling, thus illustrating the difficulty of getting estimators with good error bounds.

Charikar et. al. give a negative result in [1], where they show that for a uniform-random sample of r tuples from a table of n tuples, no distinct-value estimator can guarantee a ratio error $< O(\sqrt{n/r})$ with high probability on all inputs. We show that with block-level sampling, the guarantees that can be given are even weaker. For a block-level sample of r tuples, this lower bound can be strengthened to $O(\sqrt{nb/r})$ where b is the number of tuples per block.

THEOREM 3. *Any distinct-value estimator that examines at most R blocks from a table of N blocks, cannot guarantee a ratio error $< O(\sqrt{Nb/R})$ with high probability on all inputs, where b is the number of tuples per block.*

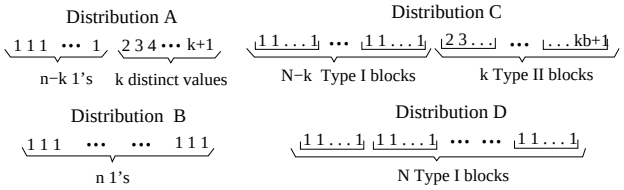

Figure 4: Negative result for distinct-value estimation

PROOF. We first review the proof of the negative result in [1]. Consider two different attribute-value distributions A and B as shown in Figure 4. Consider any distinct-value estimator that examines at most r out of the n tuples. For distribution B, the estimator shall always obtain r copies of value 1. It is shown in [1] that for distribution A, with probability at least γ, the estimator shall obtain r copies of value 1 provided:

$$k \leq \frac{n-r}{2r} \ln \frac{1}{\gamma} \tag{5}$$

In this case, the estimator cannot distinguish between distributions A and B. Let α be the value returned by the estimator in this case. This gives an error of $(k+1)/\alpha$ for distribution A, and α for distribution B. Irrespective of α, the error is at least $\sqrt{k+1}$ for one of the distributions. Choose k according to Equation 5. Then, with probability at least γ, the error is at least $O(\sqrt{n/r})$.

To extend this argument to block-level sampling, consider distributions C and D, and their layouts as shown in Figure 4. Type I blocks contain b duplicates of the value 1, while Type II blocks contain b new distinct values. Consider a distinct-value estimator that examines at most R out of N blocks. For distribution D, it always obtains R type I blocks. For distribution C, using the same argument as above, if $k \leq \frac{N-R}{2R} \ln \frac{1}{\gamma}$, then with probability at least γ, the estimator shall obtain R type I blocks. Thus, the estimator cannot distinguish between distributions C and D in this case, and must have an error of at least $\sqrt{kb+1}$ for one of the distributions. Thus, with probability at least γ, the error is at least $O(\sqrt{Nb/R})$. Hence, it is not possible to guarantee an error $< O(\sqrt{Nb/R})$ with high probability on all inputs. □

The above lower-bound notwithstanding, it is still instructive to evaluate the performance of estimators for more general layouts in terms of the ratio-error metric. We give a qualitative evaluation of the performance of COLLAPSE by comparing it with the approach of estimating distinct values using S_{unf}, i.e., a uniform-random sample of the collapsed table T_{coll}. We assume that the same distinct-value estimator is used in each case, and is of the form $\hat{D} = d + K \cdot f_1$ as in Section 4.2. Theorem 2 says that both approaches will have the same bias. However, the error of COLLAPSE may be higher. This is because although the expected value of f_1 is the same in both S_{coll} and S_{unf} (recall Lemma 1), the variance of f_1 in S_{coll} may be higher than in S_{unf}. For example, for the layout C shown in Figure 4, f_1 in S_{coll} can only take on values which are multiples of b (assuming > 1 Type I blocks are picked up in the sample). On the other hand, f_1 in S_{unf} can take on any value from 0 to kb. This larger variance leads to a higher average error for COLLAPSE.

The layouts in which the variance of f_1 in S_{coll} (and hence the average error of COLLAPSE) is higher, are those in which the number of distinct values in blocks varies widely across blocks. Based on this intuition, we introduce a *quantitative measure* for the "badness" of a layout for distinct-value estimation with block-level samples. We denote this measure as *DV_Badness*. Let d_j be the number of distinct values in the j^{th} block. Let μ be the mean, and σ be the standard deviation of d_j's ($j = 1, \ldots, N$). We define *DV_Badness* as the coefficient of variation of the d_j's, i.e., σ/μ. The higher the value of *DV_Badness*, the higher the error of COLLAPSE.

Notice that *Hist_Badness* and *DV_Badness* are different measures. Hence the layouts which are bad for histogram construction are not necessarily bad for distinct-value estimation, and vice-versa. For example, while *Hist_Badness* is maximized when the table is fully clustered, it is not so with *DV_Badness*. In fact, even when the table is fully clustered, COLLAPSE may perform very well, as long as the number of distinct values across blocks does not vary a lot (so that *DV_Badness* is still small).

5. EXPERIMENTS

In this section, we provide experimental validation of our proposed approaches. We have prototyped and experimented with our algorithms on Microsoft SQL Server running on an Intel 2.3 GHz processor with 1GB RAM.

For histogram construction, we compare our adaptive two-phase approach 2PHASE, against the iterative approach DOUBLE. We experimented with both the maxdiff bucketing algorithm (as implemented in SQL Server) as well as the equi-depth bucketing algorithm. The version of DOUBLE which we use for comparison is not exactly the same as described in [2], but an adaption of the basic idea therein to work with maxdiff as well as equi-depth histograms, and uses the variance-error metric instead of the max-error metric.

For distinct-value estimation, we compare our proposed approach COLLAPSE, with the naïve approach TAKEALL, and the ideal (but impractical) approach UNIFORM. For UNIFORM, we used a uniform-random sample of the same size as the block-level sample used by COLLAPSE. We experimented using both the HYBSKEW [7], and the AE [1] estimators.

Our results demonstrate for varying data distributions and layouts:

- For both maxdiff and equi-depth histograms, 2PHASE accurately predicts the sample size required, and is considerably faster than DOUBLE.
- For distinct value estimation, COLLAPSE produces much more accurate estimates than those given by TAKEALL, and almost as good as those given by UNIFORM.
- Our quantitative measures *Hist_Badness* and *DV_Badness*, accurately reflect the performance of block-level sampling as compared to uniform-random sampling for histogram construction and distinct-value estimation respectively.

We have experimented with both synthetic and real databases.

Synthetic Databases: To generate synthetic databases with a wide variety of layouts, we adopt the following generative model: A fraction C between 0 and 1 is chosen. Then, for each distinct value in the column of interest, a fraction C of its occurrences are given consecutive tuple-ids, and the remaining $(1-C)$ fraction are given random tuple-ids. The resulting relation is then clustered on tuple-id. We refer to C as the "degree of clustering". Different values of C give us a continuum of layouts, ranging from a random layout for $C = 0$, to a fully clustered layout for $C = 1$. This is the model which was experimented with in [2]. Besides, this model captures many real-life situations in which correlations can be expected to exist in blocks, such as those described in Section 3.1.2. Our experimental results demonstrate the relationship of the degree of clustering according to our generative model (C), with the measures of badness *Hist_Badness* and *DV_Badness*.

We generated tables with different characteristics along the following dimensions: (1) Degree of clustering C varied from 0 to 1

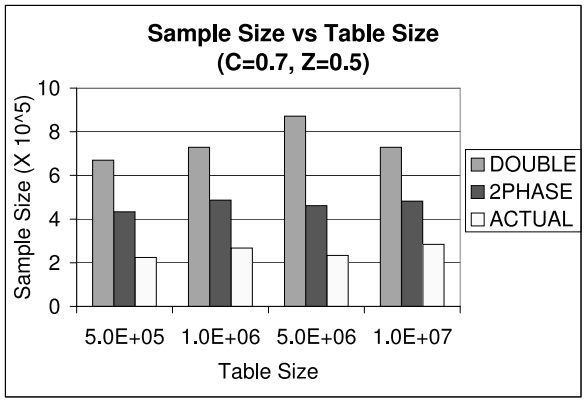

Figure 5: Effect of table size on sample size for maxdiff histograms

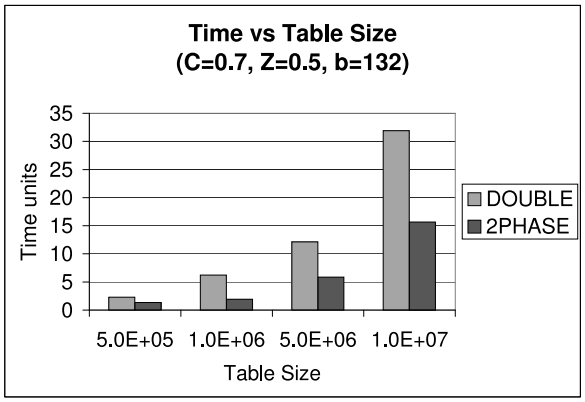

Figure 6: Effect of table size on total time for maxdiff histograms

according to our generative model, (2) Number of tuples n varied from 10^5 to 10^7, (3) Number of tuples per block b varied from 50 to 200, and (4) Skewness parameter Z varied from 0 to 2, according to the Zipfian distribution [16].

Real Databases: We also experimented with a portion of a *Home* database obtained from MSN (http://houseandhome.msn.com/). The table we obtained contained 667877 tuples, each tuple representing a home for sale in the US. The table was clustered on the *neighborhood* column. While the table had numerous other columns, we experimented with the *zipcode* column, which is expected to be strongly correlated with the neighborhood column. The number of tuples per block was 25.

5.1 Results on Synthetic Data

5.1.1 Histogram Construction

We compared 2PHASE and DOUBLE. In both approaches, we used a client-side implementation of maxdiff and equi-depth histograms [14]. We used block-level samples obtained through the sampling feature of the DBMS. Both DOUBLE and 2PHASE were started with the same initial sample size.

In our results, all quantities reported are those obtained by averaging five independent runs of the relevant algorithm. For each parameter setting, we report a comparison of the total amount sampled by 2PHASE, against that sampled by DOUBLE. We also report, the *actual* amount (denoted by ACTUAL) to be sampled to reach the desired error. This was obtained by a very careful iterative approach, in which the sample size was increased iteratively by a small amount until the error target was met. This actual size does not include the amount sampled for cross-validation. This approach is impractical due to the huge number of iterations, but reported here only for comparison purposes. We also report a comparison of the time taken by 2PHASE, against that taken by DOUBLE. The reported time[3] is a sum of the server-time spent in executing the sampling queries, and the client time spent in sorting, merging, cross-validation, and histogram construction.

We experimented with various settings of all parameters. However, due to lack of space we only report a subset of the results. We report the cases where we set the cross-validation error target at $\Delta^{req} = 0.25$, the number of buckets in the histogram at $k = 100$, and the number of tuples per block at $b = 132$. For each experiment, we provide results for only one of either maxdiff or equi-depth histograms, since the results were similar in both cases.

[3]All reported times are relative to the time taken to sequentially scan 10MB of data from disk.

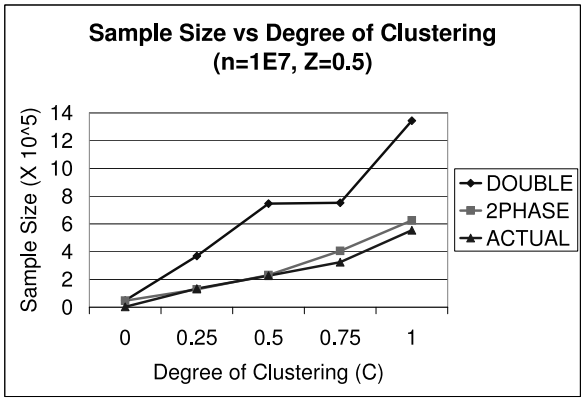

Figure 7: Effect of clustering on sample size for equi-depth histograms

Effect of n: Figure 5 shows a comparison of the amount sampled, and Figure 6 shows a time comparison for varying n, for the case of maxdiff histograms. It can be seen that the amount sampled by each approach is roughly independent of n. Also, DOUBLE substantially overshoots the required sample size (due to the last cross-validation step), whereas 2PHASE does not overshoot by as much. For n=5E5, the total amount sampled by DOUBLE exceeds the table size, but this is possible since the sampling is done in steps until the error target is met.

In terms of time, 2PHASE is found to be considerably faster than DOUBLE. Interestingly, the total time for both 2PHASE and DOUBLE increases with n even though the amount sampled is roughly independent of n. This shows that there is a substantial, fixed overhead associated with each sampling step which increases with n. This also explains why the absolute time gain of 2PHASE over DOUBLE increases with n. DOUBLE incurs the above overhead in each iteration, whereas 2PHASE incurs it only twice. Consequently, 2PHASE is much more scalable than DOUBLE.

Effect of degree of clustering: Figure 7 shows the amount sampled, and Figure 8 gives a time comparison for varying degree of clustering (C) for equi-depth histograms. Figure 8 also shows (by the dotted line) the badness measure *Hist_Badness* on a secondary axis. *Hist_Badness* was measured according to the bucket separators of the perfect histogram. Since *Hist_Badness* is maximized when the table is fully clustered, we have normalized the measure with respect to *Hist_Badness* for $C = 1$. As C increases, both *Hist_Badness*, and the required sample size increase. Thus, *Hist_Badness* is a good measure of the badness of the layout.

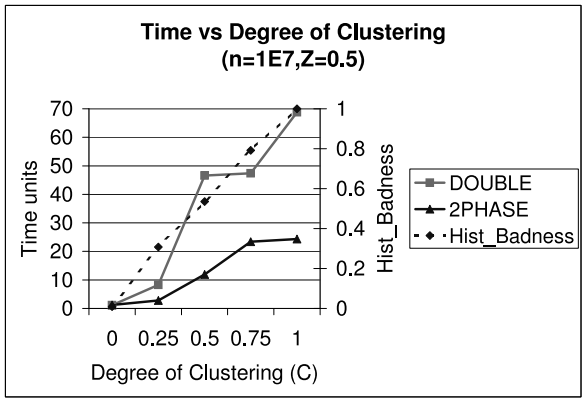

Figure 8: Effect of clustering on time for equi-depth histograms

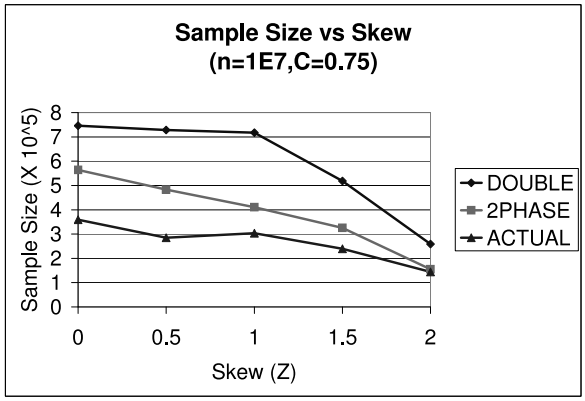

Figure 9: Effect of skew on sample size for maxdiff histograms

For $C = 0.5$, DOUBLE overshoots the required sample size almost by a factor of 4 (which is the worst case for DOUBLE). Hence, the total amount sampled becomes almost the same as that for $C = 0.75$. This is a consequence of the fact that DOUBLE picks up samples in increasingly large chunks. The time gain of 2PHASE over DOUBLE increases with C, since the latter has to go through a larger number of iterations when the required sample size is larger. The results for maxdiff histograms were similar.

Effect of Z: Figure 9 compares the amount sampled for varying skew (Z) of the distribution, for maxdiff histograms. As the skew increases, some buckets in the maxdiff histogram become very large. Consequently, a smaller sample size is required to estimate these bucket counts accurately. Thus, the required sample size decreases with skew. However, 2PHASE continues to predict the required sample size more accurately than DOUBLE. A time comparison for this experiment is omitted, as the gains of 2PHASE over DOUBLE were similar to that observed in previous experiments. Also, we omit results for equi-depth histograms, which showed very little dependence on skew.

Due to space constraints, we omit results of experimenting with varying b, k and Δ^{req}. The results in these experiments were as expected. As b increases, or k increases, or Δ^{req} decreases, the required sample size goes up (by Theorem 1). The amount by which DOUBLE overshoots 2PHASE increases. So does the time gain of 2PHASE over DOUBLE.

5.1.2 Distinct-Value Estimation

For distinct value estimation, we use the two contending estimators AE [1] and HYBSKEW [7], which have been shown to work best in practice with uniform-random samples. We consider each of these estimators with each of the three approaches— COLLAPSE, TAKEALL, and UNIFORM. We use AE_COLLAPSE to denote the COLLAPSE approach being used with the AE estimator. Other estimates are named similarly. The usability of a distinct-value estimator depends on its average ratio-error rather than on its bias (since it possible to have an unbiased estimator with arbitrarily high ratio-error). Thus, we only report the average ratio-error for each of the approaches. The average was taken over ten independent runs. In most cases, we report results only with the AE estimator, and omit those with HYBSKEW, as the trends were similar.

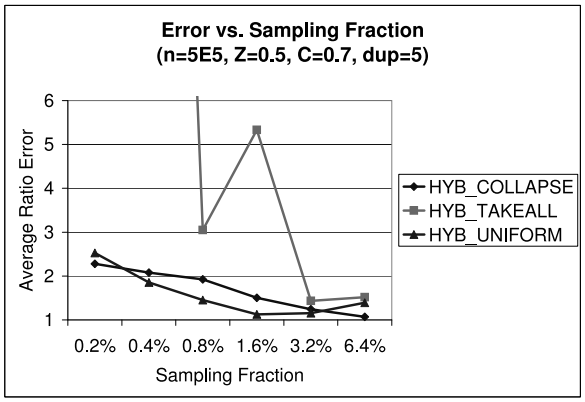

Figure 10: Variation of error with the sampling fraction for HYBSKEW

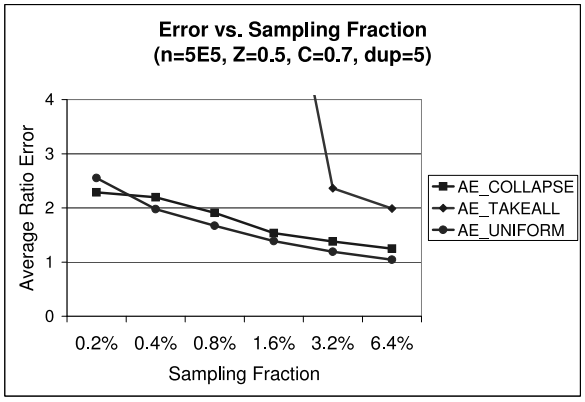

Figure 11: Variation of error with the sampling fraction for AE

For the following experiments, we added another dimension to our data generation process— the duplication factor (dup). This is the multiplicity assigned to the rarest value in the Zipfian distribution. Thus, increasing dup increases the multiplicity of each distinct value, keeping the number of distinct values constant. The number of tuples per block was again fixed at $b = 132$.

Effect of sampling fraction: Figures 10 and 11 show the error of the HYBSKEW and AE estimators respectively with the three approaches, for varying sampling fractions. With both estimators, TAKEALL leads to very high errors (as high as 200 for low sampling fractions), while COLLAPSE performs almost as well as UNIFORM for all sampling fractions.

Effect of degree of clustering: Figure 12 shows the average ratio-error of the AE estimator with the three approaches, for a fixed sampling fraction, and for varying degrees of clustering (C). As expected, the performance of UNIFORM is independent of the degree of clustering. The performance of TAKEALL degrades with increasing clustering. However, COLLAPSE continues to perform almost as well as UNIFORM even in the presence of clustering. In

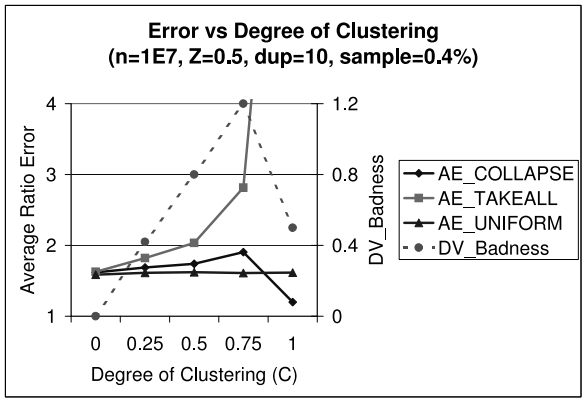

Figure 12: Effect of clustering on error for the AE estimator

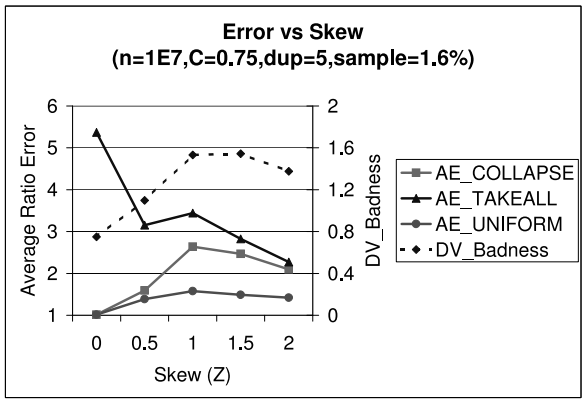

Figure 13: Effect of skew on the error for the AE estimator

Figure 12, we also show (by the dotted line) the measure of badness of the layout (*DV_Badness*), on the secondary y-axis. It can be seen that the trend in *DV_Badness* accurately reflects the performance of COLLAPSE against UNIFORM. Thus, *DV_Badness* is a good measure of the badness of the layout.

Note that unlike *Hist_Badness* (Figure 8), *DV_Badness* is not maximized when the table is fully clustered. In fact, for $C = 1$, COLLAPSE outperforms UNIFORM. This is because when the table is fully clustered (ignoring the values which occur in multiple blocks, since there are few of them), the problem of distinct-value estimation through block-level sampling can be viewed as an aggregation problem — each block has a certain number of distinct values, and we want to find the sum of these numbers by sampling a subset. Moreover, the variance of these numbers is small, as indicated by a small *Hist_Badness*. This leads to a very accurate estimate being returned by COLLAPSE. We omit the results with the HYBSKEW estimator, which were similar.

Effect of skew: Figure 13 shows the average error of AE with the three approaches, for a fixed sampling fraction, and for varying skew (Z). Here again, COLLAPSE performs consistently better than TAKEALL. We again show *DV_Badness* by the dotted line on the secondary axis. The trend in *DV_Badness* accurately reflects the error of COLLAPSE against that of UNIFORM.

Although we do not report results with HYBSKEW here, it was found that for high skew, HYB_TAKEALL actually performed consistently *better* than HYB_COLLAPSE or HYB_UNIFORM. This seems to violate our claim of COLLAPSE being a good strategy. However, at high skew, the HYBSKEW estimator itself is not very accurate, and overestimates the number of distinct values. This, combined with the tendency of TAKEALL to underestimate (due

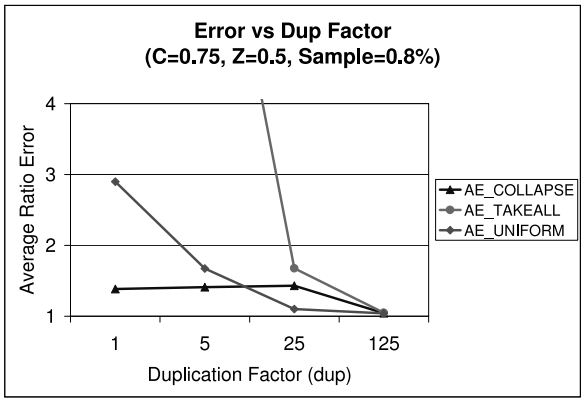

Figure 14: Effect of duplication factor on the error for the AE estimator

to its failure to recognize rare values), produced a more accurate final estimate than either COLLAPSE or UNIFORM. Thus, the good performance of HYB_TAKEALL for this case was coincidental, resulting from the inaccuracy of the HYBSKEW estimator.

Effect of bounded domain scaleup: For this experiment, the table size was increased while keeping the number of distinct values constant (by increasing dup). Figure 14 shows the average error of AE with the three approaches for a fixed sampling fraction, and for various values of dup. It can be seen that at low values of dup, TAKEALL performs very badly. However, as dup increases, almost all distinct values are picked up in the sample, and the estimation problem becomes much easier. Thus, at high values of dup, TAKEALL begins to perform well. COLLAPSE performs consistently almost as well, or better than UNIFORM.

For low values of dup, the superior performance of COLLAPSE against UNIFORM is only because the underlying AE estimator is not perfect. For low values of dup, there are a large number of distinct values in the table, and AE_UNIFORM tends to underestimate the number of distinct values. However, for AE_COLLAPSE, the value of f_1 is higher due to the collapse step. Hence the overall estimate returned is higher.

Effect of unbounded domain scaleup: For this experiment, the table size was increased while proportionately increasing the number of distinct values (keeping dup constant). Similar to the observation in [1], for a fixed sampling fraction, the average error remained almost constant for all the approaches (charts omitted due to lack of space). This is because when dup is constant, the estimation problem remains equally difficult with increasing table size.

5.2 Results on Real Data

Histogram Construction: We experimented with 2PHASE and DOUBLE on the *Home* database. The number of buckets in the histogram was fixed at $k = 100$. Figure 15 shows the amount sampled by both approaches for various error targets (Δ^{req}) for maxdiff histograms. Again, 2PHASE predicts the required size accurately while DOUBLE significantly oversamples. Note that for this database the number of tuples per block is only 25, for higher b we expect 2PHASE to perform even better than DOUBLE. Also, in this case, the sampled amounts are large fractions of the original table size (e.g., 13% sampled by 2PHASE for $\Delta^{req} = 0.2$). However, our original table is relatively small (about 0.7 million rows). Since required sample sizes are generally independent of original table sizes (e.g., see Figure 5), for larger tables the sampled fractions will appear much more reasonable. We omit the time comparisons as the differences were not substantial for this small table. Also, the results for equi-depth histograms were similar.

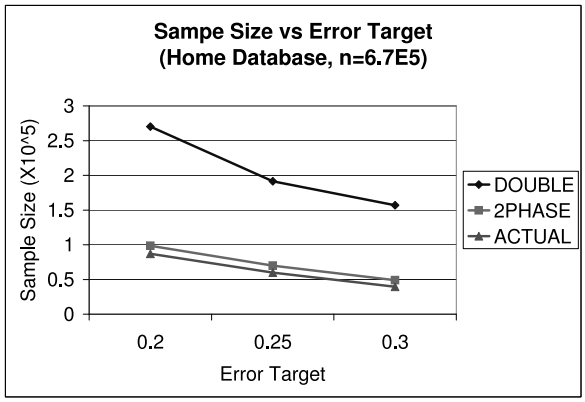

Figure 15: Variation of Sample Size with Error Target for maxdiff histograms on the *Home* database

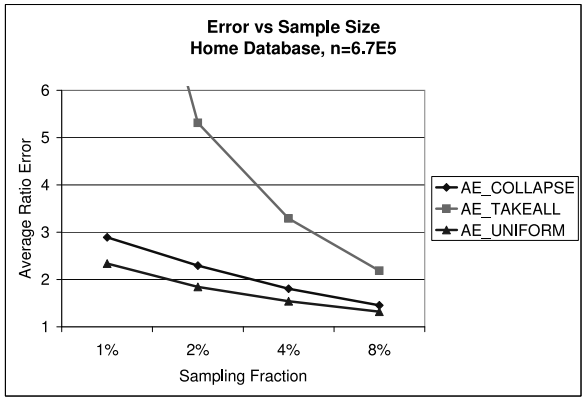

Figure 16: Distinct values estimations on the *Home* database

Distinct Value Estimation: The results of our distinct-value estimation experiments on the *Home* database are summarized in Figure 16. As expected, AE_COLLAPSE performs almost as well as AE_UNIFORM, while AE_TAKEALL performs very poorly.

6. CONCLUSIONS

In this paper, we have developed effective techniques to use block-level sampling instead of uniform-random sampling for building statistics. For histogram construction, our approach is significantly more efficient and scalable than previously proposed approaches. To the best of our knowledge, our work also marks the first principled study of the effect of block-level sampling on distinct-value estimation. We have demonstrated that in practice, it is possible to get almost the same accuracy for distinct-value estimation with block-level sampling, as with uniform-random sampling. Our results here may be of independent interest to the statistics community for the problem of estimating the number of classes in a population through cluster sampling.

Acknowledgements

We thank Peter Zabback and other members of the Microsoft SQL Server team for several useful discussions.

7. REFERENCES

[1] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. of the ACM Symp. on Principles of Database Systems*, 2000.

[2] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 436–447, 1998.

[3] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, 1977.

[4] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[5] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. of the 1997 Intl. Conf. on Very Large Data Bases*, pages 466–475, 1997.

[6] L. Goodman. On the estimation of the number of classes in a population. *Annals of Math. Stat.*, 20:572–579, 1949.

[7] P. Haas, J. Naughton, P. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of the 1995 Intl. Conf. on Very Large Data Bases*, pages 311–322, Sept. 1995.

[8] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data*, pages 341–350, 1992.

[9] W. Hou, G. Ozsoyoglu, and E. Dogdu. Error-Constrained COUNT Query Evaluation in Relational Databases. In *Proc. of the 1991 ACM SIGMOD Intl. Conf. on Management of Data*, pages 278–287, 1991.

[10] W. Hou, G. Ozsoyoglu, and B. Taneja. Statistical estimators for relational algebra expressions. In *Proc. of the 1988 ACM Symp. on Principles of Database Systems*, pages 276–287, Mar 1988.

[11] K. Burnham and W. Overton. Robust estimation of population size when capture probabilities vary among animals. *Ecology*, 60:927–936, 1979.

[12] R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. of the 1990 ACM SIGMOD Intl. Conf. on Management of Data*, pages 1–11, 1990.

[13] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. of the 1984 ACM SIGMOD Intl. Conf. on Management of Data*, pages 256–276, 1984.

[14] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*, pages 294–305, 1996.

[15] Shlosser A. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engrg. Cybernetics*, 19:97–102, 1981.

[16] G. E. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, Inc., 1949.