Scalable Histograms on Large Probabilistic Data

Mingwang Tang and Feifei Li
School of Computing, University of Utah, Salt Lake City, USA
{tang, lifeifei}@cs.utah.edu

ABSTRACT

Histogram construction is a fundamental problem in data management, and a good histogram supports numerous mining operations. Recent work has extended histograms to probabilistic data [5–7]. However, constructing histograms for probabilistic data can be extremely expensive, and existing studies suffer from limited scalability [5–7]. This work designs novel approximation methods to construct scalable histograms on probabilistic data. We show that our methods provide constant approximations compared to the optimal histograms produced by the state-of-the-art in the worst case. We also extend our methods to parallel and distributed settings so that they can run gracefully in a cluster of commodity machines. We introduced novel synopses to reduce communication cost when running our methods in such settings. Extensive experiments on large real data sets have demonstrated the superb scalability and efficiency achieved by our methods, when compared to the state-of-the-art methods. They also achieved excellent approximation quality in practice.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management—*Systems*

Keywords

histogram; scalable method; probabilistic database

1. INTRODUCTION

In many applications, uncertainty naturally exists in the data due to a variety of reasons. For instance, data integration and data cleaning systems produce fuzzy matches [10, 21]; sensor/RFID readings are inherently noisy [4, 9]. Numerous research efforts were devoted to represent and manage data with uncertainty in a probabilistic database management system [18, 21]. Many interesting mining problems have recently surfaced in the context of uncertain data e.g., mining frequent pattern and frequent itemset [1, 3, 24]. In the era of big data, along with massive amounts of data from different application and science domains, uncertainty in the data is only expected to grow with larger scale.

Histograms are important tools to represent the distribution of feature(s) of interest (e.g., income values) [15, 19]. Not surprisingly, using the possible worlds semantics [8, 21], histograms are also useful tools in summarizing and working with probabilistic data [5–7]. Given that answering queries with respect to all possible worlds is in #P-complete complexity [8], obtaining a compact synopsis or summary of a probabilistic database is of essence for understanding and working with large probabilistic data [5–7]. For example, they will be very useful for mining frequent patterns and itemsets from big uncertain data [1, 3, 24].

Cormode and Garofalakis were the first to extend the well-known V-optimal histogram (a form of bucketization over a set of one dimension values) [15], and wavelet histogram [16] to probabilistic data [6, 7], followed by the work by Cormode and Deligiannakis [5]. Note that histogram construction can be an expensive operation, even for certain data, e.g., the exact algorithm for building a V-optimal histogram is based on a dynamic programming formulation, which runs in $O(Bn^2)$ for constructing B buckets over a domain size of n [15]. Not surprisingly, building histograms on probabilistic data is even more challenging. Thus, existing methods [5–7] do not scale up to large probabilistic data, as evident from our analysis and experiments in this work.

Thus, this work investigates the problem of scaling up histogram constructions in large probabilistic data. Our goal is to explore quality-efficiency tradeoff, when such tradeoff can be analyzed and bounded in a principal way. Another objective is to design methods that can run efficiently in parallel and distributed fashion, to further mitigate the scalability bottleneck using a cluster of commodity machines.

Overview. A probabilistic database characterizes a probability distribution of an exponential number of possible worlds, and each possible world is a realization (deterministic instance) of the probabilistic database. Meanwhile, the query result on a probabilistic database essentially determines a distribution of possible query answers across all possible worlds. Given the possible worlds semantics, especially for large probabilistic data, approximate query answering based on compact synopsis (e.g., histogram) is more desirable in many cases, e.g., cost estimations in optimizers and approximate frequent items [3, 5–7, 21, 24, 26].

Conventionally, histograms on a deterministic database seek to find a set of constant bucket representatives for the

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623640 .

data distribution subject to a given space budget of buckets and an error metric. Building histograms on deterministic databases has been widely explored and understood in the literature. In probabilistic databases, building the corresponding histograms need to address the following problems: (I) how to combine the histograms on each possible world; (II) how to compute the histogram efficiently without explicitly instantiating all possible worlds.

One meaningful attempt is building histograms that seek to minimize the expected error of a histogram's approximation of item frequencies across all possible worlds, using an error metric, which was first proposed in [6, 7]. One concrete application example might be estimating the expected result size of joining two probabilistic relations based on the corresponding histograms, or evaluating queries asking for an expected value approximately.

It is *important to note that for many error metrics, this histogram is not* the same as simply building a histogram for expected values of item frequencies; and the latter always provides (much) worse quality in representing the probabilistic database with respect to a number of commonly used error metrics, as shown in [6, 7]

Based on this definition, a unified dynamic programming (DP) framework of computing optimal histograms on the probabilistic data was proposed in [6, 7] with respect to various kinds of error metrics. Specifically, for the widely used sum of square error (SSE), it costs $O(Bn^2)$ time where B is the number of buckets and n is the domain size of the data. Immediately, we see that the optimal histogram construction suffers from quadratic complexity with respect to the domain size n. For a domain of merely $100,000$ values, this algorithm could take almost a day to finish and render it unsuitable for many data sets in practice.

Summary of contributions. Inspired by these observations, we propose constant-factor approximations for histograms on large probabilistic data. By allowing approximations, we show that it is possible to allow users to adjust the efficiency-quality tradeoff in a principal manner.

We propose a novel "partition-merge" method to achieve this objective. We introduce "recursive merging" to improve the efficiency, while the histogram quality achieved will not significantly deviate from the optimal version. We also devise novel synopsis techniques to enable distributed and parallel executions in a cluster of commodity machines, to further mitigate the scalability bottleneck. To that end,

- We review the problem of histogram constructions on probabilistic data in Section 2, and highlight the limitations in the state-of-the-art.
- We design PMERGE in Section 3, which gives constant-factor approximations and scales up the histogram construction on large probabilistic data. PMERGE uses a "partition-merge" approach to realize efficiency-quality tradeoff. It also admits "recursive-merging" to allow further efficiency-quality tradeoff.
- We extend our investigation to distributed and parallel settings in Section 4, and introduce novel synopsis methods to support computation- and communication-efficient execution of our methods in distributed and parallel fashion in Section 5.
- We conduct extensive experiments on large data sets in Section 6. The results suggest that our approximation methods have achieved significant (orders of magnitude) run-time improvement compared to the state-of-the-art approach with high-quality approximation.

In addition, we survey other related works in Section 7 and conclude the paper in Section 8. *Unless otherwise specified*, proofs of theorems and lemmas were omitted due the space constraint and for brevity; they are available in Appendix B of our online technical report [25].

2. BACKGROUND AND STATE OF THE ART

Uncertain data models. Sarma *et al.* [20] describes various models of uncertainty, varying from the simplest *basic model* to the (very expensive) *complete model* that can describe any probability distribution of data instances.

Basic model is a over-simplification with no correlations. Existing work on histograms on uncertain data [5–7] adopted two popular models that extend the basic model, i.e., the *tuple model* and the *value model*, and compared their properties and descriptive abilities. The *tuple* and *value* models are two common extensions of the basic model in terms of the tuple- and attribute-level uncertainty [20], that were extensively used in the literature (see discussion in [5–7]).

Without loss of generality, we consider that a probabilistic database $\mathcal{D}$ contains one relation (table). We also concentrate on the one dimension case or one attribute of interest.

Definition 1 *The tuple model* was originally proposed in TRIO [2]. An uncertain database $\mathcal{D}$ has a set of tuples $\tau = \{t_j\}$. Each tuple t_j has a discrete probability distribution function (pdf) of the form $\langle (t_{j1}, p_{j1}), \ldots, (t_{j\ell_j}, p_{j\ell_j}) \rangle$, specifying a set of mutually exclusive *(item, probability)* pairs. Any t_{jk}, for $k \in [1, \ell_j]$, is *an item drawn from a fixed domain* and p_{jk} is the probability that t_j takes the value t_{jk} in the j^{th} row of a relation.

When instantiating this uncertain relation to a possible world W, each tuple t_j either draws a value t_{jk} with probability p_{jk} or generates no item with probability of $1 - \sum_{k=1}^{\ell_j} p_{jk}$. The probability of a possible world W is simply the multiplication of the relevant probabilities.

Definition 2 *The value model* is a sequence τ of independent tuples. Each tuple gives the *frequency distribution of a distinct item* of the form $\langle\, j : f_j = ((f_{j1}, p_{j1}), \ldots, (f_{j\ell_j}, p_{j\ell_j}))\, \rangle$. Here, j is an item drawn from a fixed domain (e.g., source IP) and its associated pdf f_j describes the distribution of j's possible frequency values.

In particular, $\Pr[f_j = f_{jk}] = p_{jk}$ where f_{jk} is a frequency value from a frequency value domain $\mathcal{V}$; f_j is subject to the constraint that $\sum_{jk} p_{jk} \leq 1$ for $k \in [1, \ell_j]$. When it is less than 1, the remaining probability corresponds that *the item's frequency is zero*. When instantiating this uncertain relation to a possible world W, for an item j, its frequency f_j either takes a frequency value f_{jk} with probability p_{jk} or takes zero as its frequency value with probability $1 - \sum_{k=1}^{\ell_j} p_{jk}$. So the probability of a possible world W is computed as the multiplication of the possibilities of f_j's taking the corresponding frequency in each tuple.

2.1 Histograms on probabilistic data

Without loss of generality, in both models, we consider the items are drawn from the integer domain $[n] = \{1, ..., n\}$ and use $\mathcal{W}$ to represent the set of all possible worlds. Let N be the size of a probabilistic database, i.e., $N = |\tau|$.

For an item $i \in [n]$, g_i *is a random variable for the distribution of* i*'s frequency over all possible worlds*, i.e,

$$g_i = \{(g_i(W), \Pr(W)) | W \in \mathcal{W}\}, \quad (1)$$

where $g_i(W)$ is item i's frequency in a possible world W and $\Pr(W)$ is the possibility of W.

Example 1 Consider an ordered domain $[n]$ with three items $\{1, 2, 3\}$ for both models, i.e., $n = 3$.

The input $\tau = \{\langle(1, \frac{1}{2}), (3, \frac{1}{3})\rangle, \langle(2, \frac{1}{4}), (3, \frac{1}{2})\rangle\}$ *in the tuple model* defines eight possible worlds:

W	$\emptyset$	1	2	3	1, 2	1, 3	2, 3	3, 3
$\Pr(W)$	$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{24}$	$\frac{1}{6}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{12}$	$\frac{1}{6}$

The input $\tau = \{\langle 1 : (1, \frac{1}{2})\rangle, \langle 2 : (1, \frac{1}{3})\rangle, \langle 3 : ((1, \frac{1}{2}), (2, \frac{1}{2}))\rangle\}$ *in the value model* defines eight possible worlds:

W	3	1, 3	2, 3	3, 3	1, 2, 3	1, 3, 3	2, 3, 3	1, 2, 3, 3
$\Pr(W)$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{12}$

Consider the *tuple model* example from above and denote the eight possible worlds (from left to right) as $W_1, \ldots, W_8$. It's easy to see that $g_3(W) = 1$ for $W \in \{W_4, W_6, W_7\}$, $g_3(W) = 2$ for $W \in \{W_8\}$ and $g_3(W) = 0$ on the rest. Thus, the frequency random variable g_3 of item 3 is $g_3 = \{(0, \frac{1}{3}), (1, \frac{1}{2}), (2, \frac{1}{6})\}$ with respect to $\mathcal{W}$ in this example. Meanwhile, it's also easy to see $g_3 = \{(1, \frac{1}{2}), (2, \frac{1}{2})\}$ over $\mathcal{W}$ in the value model example from above.

Definition 3 A *B-bucket representation* partitions domain $[n]$ into B non-overlapping consecutive buckets (s_k, e_k) for $k \in [1, B]$, where $s_1 = 1$, $e_B = n$ and $s_{k+1} = e_k + 1$. Frequencies within each bucket b_k are approximated by a single representative $\widehat{b}_k$ and we represent it as $b_k = (s_k, e_k, \widehat{b}_k)$.

The B-bucket histogram achieving the minimal SSE error for approximating a deterministic data distribution is known as the V-optimal histogram [14]. It can be found using a dynamic programming formulation in $O(Bn^2)$ time [15], where n is the domain size of the underlying data distribution. We denote this method from [15] as the OptVHist method.

To extend histogram definitions to probabilistic data, we first consider a single possible world $W \in \mathcal{W}$ for a probabilistic data set $\mathcal{D}$, where W is *a deterministic data set*. Hence, the frequency vector of W is given by $G(W) = \{g_1(W), ..., g_n(W)\}$ (recall that $g_i(W)$ is item i's frequency in W). Given a B-bucket representation for approximating $G(W)$, the SSE of a bucket b_k in the world W is given as: $SSE(b_k, W) = \sum_{j=s_k}^{e_k} (g_j(W) - \widehat{b}_k)^2$. The SSE of the B-bucket representation in W is simply $\sum_{k=1}^{B} SSE(b_k, W)$.

Cormode and Garofalakis have extended B-bucket histogram to probabilistic data [6,7] by asking for the minimal expected SSE. Formally,

Definition 4 Given the (uncertain) frequency sequence of random variables $\{g_1, \ldots, g_n\}$ as defined in (1), the problem seeks to construct a B-bucket representation (typically $B \ll n$) such that the expected SSE over all possible worlds is minimized, i.e., the histogram with the value given by:

$$\mathcal{H}(n, B) = \min\{\mathrm{E}_{\mathcal{W}}\left[\sum_{k=1}^{B}\sum_{j=s_k}^{e_k}(g_j - \widehat{b}_k)^2\right]\} \quad (2)$$

In (2), the expectation of the sum of bucket errors is equal to the sum of expectations of bucket errors [6,7], i.e.,

$$\mathrm{E}_{\mathcal{W}}\left[\sum_{k=1}^{B}\sum_{j=s_k}^{e_k}(g_j - \widehat{b}_k)^2\right] = \sum_{k=1}^{B}\mathrm{E}_{\mathcal{W}}\left[\sum_{j=s_k}^{e_k}(g_j - \widehat{b}_k)^2\right]. \quad (3)$$

Consequently, the optimal histogram could be derived by a dynamic programming formulation as follows:

$$\mathcal{H}(i, j) = \min_{1 \le \ell < i} \mathcal{H}(\ell, j-1) + \min_{\widehat{b}}(\ell + 1, i, \widehat{b}), \quad (4)$$

where $\mathcal{H}(i, j)$ represents the minimal error from the optimal j-buckets histogram on interval $[1, i]$; $\min_{\widehat{b}}(\ell + 1, i, \widehat{b})$ is the minimal bucket error for the bucket spanning the interval $[\ell + 1, i]$ using a single representative value $\widehat{b}$.

Previous work [6, 7] showed that the cost of the optimal histogram is $O(Bn^2)$ and $\min_{\widehat{b}}(\ell+1, i, \widehat{b})$ could be computed in constant time using several precomputed prefix-sum arrays which we will describe in the following subsection. We dub this state-of-art method from [7] *the* OptHist *method.*

2.2 Efficient computation of bucket error

Cormode and Garofalakis [7] show that, for SSE, the minimal error of a bucket $b = (s, e, \widehat{b})$ is achieved by setting the representative $\widehat{b} = \frac{1}{e-s+1}\mathrm{E}_{\mathcal{W}}\left[\sum_{i=s}^{e} g_i\right]$. The corresponding bucket error is given by:

$$SSE(s, e, \widehat{b}) = \sum_{i=s}^{e}\mathrm{E}_{\mathcal{W}}[g_i^2] - \frac{1}{e-s+1}\mathrm{E}_{\mathcal{W}}[\sum_{i=s}^{e} g_i]^2. \quad (5)$$

In order to answer the $\min_{\widehat{b}}(s, e, \widehat{b})$ query in (4) for any (s, e) values in constant time, prefix-sum arrays of $\mathrm{E}_{\mathcal{W}}[g_i^2]$ and $\mathrm{E}_{\mathcal{W}}[g_i]$ in equation (5) are precomputed as follows (details can be found in [7]):

$$A[e] = \sum_{i=1}^{e}\mathrm{E}_{\mathcal{W}}[g_i^2] = \sum_{i=1}^{e}(\mathrm{Var}_{\mathcal{W}}[g_i] + \mathrm{E}_{\mathcal{W}}[g_i]^2) \quad B[e] = \sum_{i=1}^{e}\mathrm{E}_{\mathcal{W}}[g_i] \quad (6)$$

tuple model: $\mathrm{E}_{\mathcal{W}}[g_i] = \sum_{t_j \in \tau}\Pr[t_j = i]$ and $\mathrm{Var}_{\mathcal{W}}[g_i] = \sum_{t_j \in \tau}\Pr[t_j = i](1 - \Pr[t_j = i])$.

value model: $\mathrm{E}_{\mathcal{W}}[g_i] = \sum_{v_j \in \mathcal{V}} v_j \Pr[g_i = v_j]$ and $\mathrm{Var}_{\mathcal{W}}[g_i] = \sum_{v_j \in \mathcal{V}}(v_j - \mathrm{E}_{\mathcal{W}}[g_i])^2 \Pr[g_i = v_j]$

Set $A[0] = B[0] = 0$, then the minimal SSE $\min_{\widehat{b}}(s, e, \widehat{b})$ for both models is computed as:

$$A[e] - A[s-1] - \frac{(B[e] - B[s-1])^2}{e - s + 1}.$$

In both models, in addition to the $O(Bn^2)$ cost as shown in last subsection, it also takes $O(N)$ cost to compute the A, B arrays ($N = |\tau|$, number of probabilistic tuples).

3. APPROXIMATE HISTOGRAMS

The state-of-the-art OptHist method from [7] is clearly not scalable, when given larger domain size.

A baseline method. A natural choice is to consider computing a B-bucket histogram for the *expected frequencies of all items*. Note that this histogram is not the same as the desired histogram as defined in equation (2) and (3) since in general $\mathrm{E}[f(X)]$ does not equal $f(\mathrm{E}[X])$ for arbitrary function f and random variable X.

However, we can show in our histogram, the SSE error of a bucket $[s, e]$ using $\widehat{b}$ as its representative is:

$$SSE(s, e, \widehat{b}) = \mathrm{E}_{\mathcal{W}}[\sum_{i=s}^{e}(g_i - \widehat{b})^2] = \sum_{i=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i^2] - 2\mathrm{E}_{\mathcal{W}}[g_i]\widehat{b} + \widehat{b}^2).$$

On the other hand, if we build a B-bucket histogram over the expected frequencies of all items, the error of a bucket $[s, e]$ using $\bar{b}$ as its representative is:

$$SSE(s,e,\bar{b}) = \sum_{i=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i]-\bar{b})^2 = \sum_{i=s}^{e}((\mathrm{E}_{\mathcal{W}}[g_i])^2-2\mathrm{E}_{\mathcal{W}}[g_i]\bar{b}+\bar{b}^2).$$

When using the same bucket configurations (i.e., the same boundaries and $\widehat{b} = \bar{b}$ for every bucket), the two histograms above differ by $\sum_{j=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i^2]-(\mathrm{E}_{\mathcal{W}}[g_i])^2) = \sum_{j=s}^{e}\mathrm{Var}_{\mathcal{W}}[g_i]$ on a bucket $[s,e]$. Hence, the overall errors of the two histograms differ by $\sum_{i\in[n]}\mathrm{Var}_{\mathcal{W}}[g_i]$ which is a constant. Given this and computing the expected frequencies of all items can be done in $O(N)$ time, computing the optimal B-bucket histogram for them (now a deterministic frequency vector) still requires the OptVHist method from [15], taking $O(Bn^2)$ for a domain of size n, which still suffers the same scalability issue.

A natural choice is then to use an approximation for the B-bucket histogram on expected frequencies (essentially a V-optimal histogram), as an approximation for our histogram. The best approximation for a V-optimal histogram is an $(1+\varepsilon)$-approximation [23] (in fact, to the best of our knowledge, it is the only method with theoretical bound on approximation quality). But when using approximations, one cannot guarantee that the same bucket configurations will yield the same approximation bound with respect to both histograms. So its theoretical guarantee is no longer valid with respect to our histogram. Nevertheless, it is worth comparing to this approach as a baseline method, which is denoted as the *EF-Histogram* method.

3.1 The PMerge method

Hence, we search for novel approximations that can provide error guarantees on the approximation quality and also offer quality-efficiency tradeoff, for the histograms from [6,7] as defined in (2). To that end, we propose a constant approximation scheme, PMerge, by leveraging a "partition-merge" principle. It has a *partition phase* and a *merge phase*.

Partition. The *partition phase* partitions the domain $[n]$ into m equally-sized sub-domains, $[s_1,e_1],\ldots,[s_m,e_m]$ where $s_1 = 1, e_m = n$ and $s_{k+1} = e_k + 1$. For the kth sub-domain $[s_k,e_k]$, we compute the A,B arrays on this domain as A_k, B_k for $k \in [1,m]$. A_k and B_k are computed using $[s_k,e_k]$ as an input domain and equation (6) for the value and the tuple models respectively,

Next, for each sub-domain $[s_k,e_k]$ ($k \in [1,m]$), we apply the OptHist method from [7] (as reviewed in Section 2.1) over the A_k, B_k arrays to find the *local optimal B-buckets histogram* for the kth sub-domain. The partition phase produces m local optimal B-bucket histograms, which lead to mB buckets in total.

Merge. The goal of the merge phase is to merge the mB buckets from the partition phase into optimal B buckets in terms of the SSE error using one merging step. To solve this problem, naively, we can view an input bucket $b = (s,e,\widehat{b})$ as having $(e-s+1)$ items with identical frequency value $\widehat{b}$. Then, our problem reduces to precisely constructing an V-optimal histogram instance [15]. But the cost will be $O(B(\sum_{i=1}^{mB}(e_i - s_i + 1))^2)$ using the OptVHist method, which is simply $O(Bn^2)$.

A critical observation is that a bucket $b = (s,e,\widehat{b})$ can also be viewed as a *single weighted frequency* $\widehat{b}$ with a weight of $(e-s+1)$, such that we can effectively reduce the domain size while maintaining the same semantics. Formally, let $Y = mB$. A weighted frequency vector $\{f_1, f_2, \ldots, f_Y\}$ on an ordered domain $[Y]$ has a weight w_i for each f_i. It implies w_i items with a frequency f_i at i. The weighted version of the V-optimal histogram seeks to construct a B-bucket histogram such that the SSE between these buckets and the input weighted frequency vector is minimized. This problem is the same as finding:

$$\mathcal{H}_w(Y,B) = \min\{\sum_{k=1}^{B}\sum_{j=s_k}^{e_k} w_j(f_j - \widehat{b}_k)^2\},$$

where $s_1 = 1$ and $e_B = Y$. The optimal B buckets can be derived by a similar dynamic programming formulation as that shown in equation (4). The main challenge is to compute the optimal one-bucket $\min_{\widehat{b}}(s,e,\widehat{b})$ for any interval $[s,e]$ now in the weighted case. We show in Appendix A how to do this efficiently using several prefix sum arrays.

Thus the weighted optimal B-bucket histogram can be derived by filling a $Y \times B$ matrix, and each cell (i,j) takes $O(Y)$ time. Thus, the weighted B-bucket histogram is computed in $O(BY^2) = O(m^2B^3)$ time, which is much less than $O(Bn^2)$ since both B and m are much smaller than n.

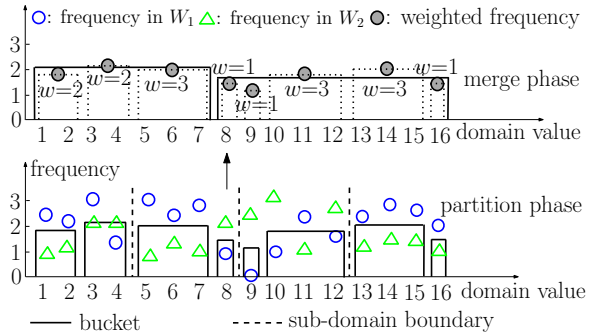


Figure 1: An example of PMerge: $n = 16, m = 4, B = 2$.

An example. An example of PMerge is given in Figure 1, where $n = 16$, $B = 2$, and $m = 4$. To ensure clarity, we show only two possible worlds W_1 (blue circle) and W_2 (green triangle) from the set of possible worlds $\mathcal{W}$ of this database. In the partition phase, each sub-domain of size 4 is approximated by 2 local optimal buckets. In total, the partition phase has produced 8 buckets in Figure 1. In the merge phase, each input bucket maps to a weighted frequency as discussed above. For example, the first bucket covering frequencies in $[1,2]$ represents a weighted frequency of 1.8 with weight 2. These 8 buckets were merged into two buckets as the final output.

Complexity analysis. In the partition phase, it takes linear time to compute the corresponding A_k, B_k arrays within each sub-domain $[s_k,e_k]$ for $k \in [1,m]$, following the results from [7]. The size of sub-domain $[s_k,e_k]$ is roughly n/m for $k \in [1,m]$. It takes $O(Bn^2/m^2)$ to run the OptHist method on A_k, B_k to find the kth local optimal B-bucket histogram. Next, the merge phase takes only $O(B^3m^2)$ time as analyzed above. Hence,

Lemma 1 PMerge *takes* $O(N + Bn^2/m + B^3m^2)$.

3.2 Approximation quality

In order to evaluate the absolute value of the histogram approximation error, we adopt the ℓ_2 distance (square root of SSE error) between the data distribution and the histogram synopsis. Next, we show the approximation quality of PMerge compared to the *optimal* B-bucket histogram found by OptHist in terms of the ℓ_2 distance.

Theorem 1 *Let $\|\mathcal{H}(n,B)\|_2$ and $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2$ be the ℓ_2 norm of the SSE error of B-bucket histogram produced by* OPTHIST *and* PMERGE *respectively on domain $[n]$. Then, $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2 < 3.17 \cdot \|\mathcal{H}(n,B)\|_2$.*

3.3 Recursive PMERGE

Note that the problem size of mB in the merge phase of PMERGE may still be too large to be handled efficiently by a DP method. Fortunately, we can further improve the efficiency by doing "recursive merging" as follows.

First of all, the partition phase will partition the input domain into m^ℓ equal-sized sub-domains, instead of only m sub-domains, for some integer ℓ (user specified).

The merge phase now *recursively* merges the $m^\ell B$ buckets from the partition phase into B buckets using ℓ iterations. Each iteration reduce the number of input buckets by a factor of m by applying a sequence of *merging steps*. Specifically, each merging step merges mB consecutive buckets (from left to right) from the current iteration into B buckets in the next iteration, which is done using the same merging step from the standard PMERGE method (i.e., using the weighted B-bucket histogram idea). We dub the recursive PMERGE methods RPMERGE.

Extending the analysis from Lemma 1 and Theorem 1 gives the following result, w.r.t the ℓ_2 norm of the SSE:

Theorem 2 *Using $O(N + B\frac{n^2}{m^\ell} + B^3\sum_{i=1}^{\ell} m^{(i+1)})$ time, the* RPMERGE *method gives a 3.17^ℓ approximation of the optimal B-bucket histogram found by* OPTHIST.

It is important to note that the approximation bounds in both Theorems 1 and 2 reflect the *worst-case analysis*. The extreme cases leading to the worst-case bounds are almost impossible in real data sets. In practice, PMERGE and its recursive version RPMERGE always provide *(very) close to optimal approximation quality* (much better than what these worst-case bounds indicate), as shown in our experiments.

4. DISTRIBUTED AND PARALLEL PMERGE

PMERGE allows efficient execution in a distributed and parallel framework. In the partition phase, each sub-domain can be handled independently in parallel.

The recursive PMERGE offers even more venues for parallelism. In this case, *its merge phase* can also run in a distributed and parallel fashion, since each merging step from every iteration can be processed independently.

Next, we'll address the challenge on computing the local A_k, B_k arrays efficiently for each sub-domain $[s_k, e_k]$ in a distributed and parallel setting . For both models, we assume that the underlying probabilistic database has been split into β chunks $\{\tau_1, \ldots, \tau_\beta\}$ and stored in a distributed file system (DFS). It is important to note that the input data is not necessarily sorted by the values of the items when stored into chunks in a DFS.

4.1 The partition phase in the value model

Recall that in the value model, f_i is a pdf describing item i's possible frequency values and their associated probabilities. We first show that:

Lemma 2 *In the value model, $\Pr[g_i = v] = \Pr[f_i = v]$ for any frequency value $v \in \mathcal{V}$ ($\mathcal{V}$ is the domain of all possible frequency values).*

Lemma 2 and equation (6) imply that:

Lemma 3 *The A, B arrays for the value model also equal: $A[j] = \sum_{i=1}^{j} \mathrm{E}[f_i^2]$, $B[j] = \sum_{i=1}^{j} \mathrm{E}[f_i]$.*

Without loss of generality, we assume β "data nodes (aka processes)" to consume the input data chunks, and also m "aggregate nodes/processes" to produce the local optimal B-bucket histograms. Each data chunk is processed by one data node in parallel. Each data node produces m partitions, each of which corresponds to a sub-domain of size (roughly) n/m, using a partition function $h : [n] \to [m]$, $h(i) = (\lceil i/\lceil n/m \rceil \rceil)$.

The ℓth data node processing chunk τ_ℓ reads in tuples in τ_ℓ in a streaming fashion. For each incoming tuple (i, f_i) found in τ_ℓ, it computes two values $(\mathrm{E}[f_i], \mathrm{E}[f_i^2])$. It then writes a key-value pair $(i, (\mathrm{E}[f_i], \mathrm{E}[f_i^2]))$ to the $h(i)$th partition. The $h(i)$th aggregate node will collect the $h(i)$th partitions from all β data nodes, the union of which forms the $h(i)$th sub-domain of the entire data.

Thus, the kth ($k \in [1, m]$) aggregate node will have all the key-value pairs $(i, (\mathrm{E}[f_i], \mathrm{E}[f_i^2]))$ for all $i \in [s_k, e_k]$ in the kth sub-domain, if item i exists in the database; otherwise it simply produces a $(i, (0, 0))$ pair for such $i \in [s_k, e_k]$.

That said, the kth aggregate node can easily compute the A_k, B_k arrays for the kth sub-domain using Lemma 3. It then uses the OPTHIST method on A_k, B_k to produce the kth local optimal B-bucket histogram. Clearly, all m aggregate nodes can run independently in parallel.

4.2 The partition phase in the tuple model

In the tuple model, the tuples needed to compute $\mathrm{Var}_{\mathcal{W}}[g_i]$ and $\mathrm{E}_{\mathcal{W}}[g_i]$ for each item i are distributed over β tuple chunks. Hence, we rewrite equation (6) for computing A, B arrays in the tuple model as follows:

Lemma 4 *The A, B arrays in the tuple model can also be computed as:*

$$A[j] = \sum_{i=1}^{j}\Big(\sum_{\ell=1}^{\beta} \mathrm{Var}_{\mathcal{W},\ell}[g_i] + \Big(\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i]\Big)^2\Big) \quad B[j] = \sum_{i=1}^{j}\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i],$$

where $\mathrm{Var}_{\mathcal{W},\ell}[g_i] = \sum_{t\in\tau_\ell} \Pr[t = i](1 - \Pr[t = i])$ and $\mathrm{E}_{\mathcal{W},\ell}[g_i] = \sum_{t\in\tau_\ell} \Pr[t = i]$.

A similar procedure as that described for the value model could then be applied. The difference is that the ℓth data node processing chunk τ_ℓ emits a key-value pair $(i, (\mathrm{E}_{\mathcal{W},\ell}[g_i], \mathrm{Var}_{\mathcal{W},\ell}[g_i]))$ instead, for *each distinct item i from the union of all possible choices of all tuples in τ_ℓ*. Thus, the kth aggregate node will reconstruct A_k, B_k arrays according to Lemma (4) and then use the OPTHIST method on A_k, B_k arrays to produce the local optimal B-bucket histogram for the kth sub-domain in the partition phase.

4.3 Recursive PMERGE and other remarks

For RPMERGE, we carry out the partition phase for each model using the method from Section 4.1 and Section 4.2 respectively. In the merge phase, we can easily invoke multiple independent nodes/processes to run all merging steps in one iteration in parallel. In the following, we denote the distributed and parallel PMERGE and RPMERGE methods as parallel-PMERGE and parallel-RPMERGE respectively.

5. PARALLEL-PMERGE WITH SYNOPSIS

A paramount concern in distributed computation is the communication cost. The parallel-PMERGE method may incur high communication cost for large domain size.

This cost is $O(n)$ in the value model. Given a set τ of tuples in a value model database with size $N = |\tau|$; τ is stored in β distributed chunks in a DFS. Each tuple will produce a key-value pair to be emitted by one of the data nodes. In the worst case $N = n$ (one tuple for each item of the domain), thus $O(n)$ cost. On the other hand, this cost is $O(\beta n)$ in the tuple mode. The worst case is when possible choices from all tuples in every distributed tuple chunk have covered all distinct items from the domain $[n]$.

There are only $O(Bm)$ bytes communicated in the merge phase of parallel-PMERGE for both models, where every aggregate node sends B buckets to a single node for merging. Thus, the communication cost of parallel-PMERGE is dominated by the partition phase.

We present novel synopsis to address this issue. The key idea is to approximate the A_k, B_k arrays at the kth aggregate node ($k \in [1, m]$) with unbiased estimators $\widehat{A}_k, \widehat{B}_k$ constructed by *either samples or sketches* sent from the data nodes. Since parallel-PMERGE and parallel-RPMERGE share the same partition phase, hence, the analysis above and the synopsis methods below *apply to both methods.*

5.1 Sampling methods for the value model

The VS method. One way of interpreting $\mathrm{E}[f_i^2]$ and $\mathrm{E}[f_i]$ is treating each of them as *an count of item* i in the arrays A_k and B_k respectively. Then $A_k[j]$ and $B_k[j]$ in Lemma 3 can be interpreted as *the rank* of j, i.e. the number of appearance of items from $[s_k, e_k]$ that are less than or equal to j in array A_k, B_k respectively. Using this view, we show how to construct an estimator $\widehat{B}_k[j]$ with the value model sampling method VS. The construction and results of $\widehat{A}_k[j]$ are similar.

Considering the ℓth data node that processes the ℓth tuple chunk τ_ℓ, we first define $T1(i, \ell) = \mathrm{E}[f_i^2]$ and $T2(i, \ell) = E[f_i]$ respectively if $(i, f_i) \in \tau_\ell$; otherwise we assign them as 0. We then define $A_{k,\ell}, B_{k,\ell}$ as follows, for every $k \in [1, m]$:

$$A_{k,\ell}[j] = \sum_{i=s_k}^{j} T1(i, \ell), B_{k,\ell}[j] = \sum_{i=s_k}^{j} T2(i, \ell), \text{ for any } j \in [s_k, e_k].$$

Using τ_ℓ, the ℓth data node can easily compute $A_{k,\ell}, B_{k,\ell}$ locally for all k and j values. It's easy to get the following results at *the kth aggregate node* for any $j \in [s_k, e_k]$:

$$A_k[j] = \sum_{\ell=1}^{\beta} A_{k,\ell}[j], B_k[j] = \sum_{\ell=1}^{\beta} B_{k,\ell}[j], \text{ for any } k \in [1, m]. \quad (7)$$

We view $B_{k,\ell}[j]$ as the *local rank* of j from τ_ℓ at the ℓth data node. By (7), $B_k[j]$ is simply *the global rank of* j that equals the sum of all local ranks from β nodes. We also let $M_k = \sum_{j=s_k}^{e_k} \mathrm{E}[f_j]$.

For every tuple (i, f_i) from τ_ℓ, data node ℓ unfolds (conceptually) $\mathrm{E}[f_i]$ copies of i, and samples each i independently with probability $p = \min\{\Theta(\sqrt{\beta}/\varepsilon M_k), \Theta(1/\varepsilon^2 M_k)\}$. If a copy of i is sampled, it is added to a sample set $S_{k,\ell}$ where $k = h(i)$, using the hash function in Section 4.1. If c_i copies of i are sampled, we add $(i, 1), \ldots, (i, c_i)$ into $S_{k,\ell}$. The pairs of values in $S_{k,\ell}$ are sorted by the item values from the first term, and ties are broken by the second term. Data node ℓ sends $S_{k,\ell}$ to the kth aggregate node for $k \in [1, m]$.

We define the *rank* of a pair (i, x) in $S_{k,\ell}$ as the number of pairs ahead of it in $S_{k,\ell}$, denoted as $r((i, x))$. For any $j \in [s_k, e_k]$ and $\ell \in [1, \beta]$, aggregate node k computes an estimator $\widehat{B}_{k,\ell}[j]$ for the local rank $B_{k,\ell}[j]$ as: $\widehat{B}_{k,\ell}[j] = r((j, c_j))/p + 1/p$, if item j is present in $S_{k,\ell}$.

If an item $j \in [s_k, e_k]$ is not in $S_{k,\ell}$, let y be the predecessor of j in $S_{k,\ell}$ in terms of *item values*, then $\widehat{B}_{k,\ell}[j] = \widehat{B}_{k,\ell}[y] + 1/p$. If no predecessor exists, then $\widehat{B}_{k,\ell}[j] = 0$.

It then estimates the global rank $B_k[j]$ for $j \in [s_k, e_k]$ as:

$$\widehat{B}_k[j] = \sum_{\ell=1}^{\beta} \widehat{B}_{k,\ell}[j]. \quad (8)$$

Lemma 5 *$\widehat{B}_k[j]$ in (8) is an unbiased estimator of $B_k[j]$ and $\mathrm{Var}[\widehat{B}_k[e]]$ is $O((\varepsilon M_k)^2)$.*

The communication cost is $\sum_{\ell,j} p = O(\min\{\sqrt{\beta}/\varepsilon, 1/\varepsilon^2\})$ for $\ell \in [1, \beta]$ and $j \in [s_k, e_k]$ for aggregate node k in the worst case. Hence, the total communication cost in the partition phase of PMERGE with VS is $O(\min\{m\sqrt{\beta}/\varepsilon, m/\varepsilon^2\})$. Note that $\{M_1, \ldots, M_m\}$ can be easily precomputed in $O(m\beta)$ communication cost.

5.2 Sketching methods for the tuple model

The TS (tuple model sketching) method. Observe that we can rewrite equations in Lemma 4 to get:

$$A_k[j] = \sum_{\ell=1}^{\beta} \sum_{i=s_k}^{j} \mathrm{Var}_{\mathcal{W},\ell}[g_i] + \sum_{i=s_k}^{j} (\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i])^2$$

$$B_k[j] = \sum_{\ell=1}^{\beta} \sum_{i=s_k}^{j} \mathrm{E}_{\mathcal{W},\ell}[g_i]. \quad (9)$$

We can view $\sum_{i=s_k}^{j} \mathrm{Var}_{\mathcal{W},\ell}[g_i]$ and $\sum_{i=s_k}^{j} \mathrm{E}_{\mathcal{W},\ell}[g_i]$ as a local rank of j in a separate local array computed from τ_ℓ. Similarly, estimation of the global rank, i.e., the first term of $A_k[j]$ and $B_k[j]$ in (9), can be addressed by the VS method.

The challenge is to approximate $\sum_{i=s_k}^{j} (\mathrm{E}_{\mathcal{W}}[g_i])^2$, the second term of $A_k[j]$ in (9). It is the second frequency moment (F_2) of $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_j]\}$. Given that each $\mathrm{E}_{\mathcal{W}}[g_i]$ is a distributed sum and j varies over $[s_k, e_k]$, we actually need a *distributed method* to answer a *dynamic F_2 (energy) range query* approximately on a sub-domain $[s_k, e_k]$.

The key idea is to build AMS sketches [17] for a set of *intervals from a carefully constructed binary decomposition* on each sub-domain locally at every data node.

For a sub-domain $[s_k, e_k]$ at the kth aggregate node, let $M_k'' = \sum_{i=s_k}^{e_k} (\mathrm{E}_{\mathcal{W}}[g_i])^2$. The leaf-level of the binary decomposition partitions $[s_k, e_k]$ into $1/\varepsilon$ intervals, where each interval's F_2 equals $\varepsilon M_k''$. An index-level (recursively) concatenates every two consecutive intervals from the level below to form a new interval (thus, the height of this binary decomposition is $O(\log\lceil\frac{1}{\varepsilon}\rceil)$. Figure 2(a) illustrates this idea.

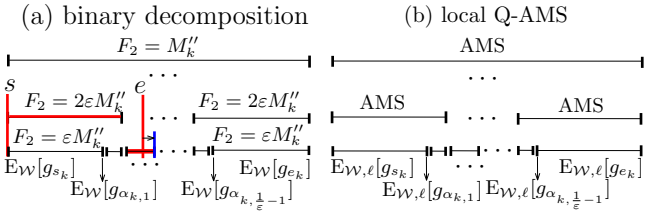


Figure 2: Binary decomposition and local Q-AMS.

Once the $(\frac{1}{\varepsilon} - 1)$ partition boundaries $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ at the leaf-level were found, aggregate node k sends them to all β data nodes. Each data node builds a set of AMS sketches, one for each interval from the binary decomposition (of all levels), over its local data. We denote it as the local *Q-AMS sketch* (Queryable-AMS).

In other words, data node ℓ builds these AMS sketches using $\{\mathrm{E}_{\mathcal{W},\ell}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W},\ell}[g_{e_k}]\}$ as shown in Figure 2(b). Then data node ℓ sends its Q-AMS sketch for $[s_k, e_k]$ to the kth aggregate node, which combines β local Q-AMS sketches into a *global Q-AMS* sketch for the kth sub-domain $[s_k, e_k]$, leveraging on the linearly-mergeable property of each individual AMS sketch [12, 17]. The global Q-AMS sketch is equivalent to a Q-AMS sketch that is built from $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_{e_k}]\}$ directly; recall that $\mathrm{E}_{\mathcal{W}}[g_i] = \sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i]$.

For a query range (s, e), $s_k \le s < e \le e_k$, we find the intervals that form the canonical cover of (s, e) in the global Q-AMS sketch, and approximate $(\mathrm{E}_{\mathcal{W}}[g_s])^2 + \cdots + (\mathrm{E}_{\mathcal{W}}[g_e])^2$ by the summation of the F_2 approximations (from the AMS sketches) of these intervals. If (s, e) is not properly aligned with an interval at the leaf-level of an Q-AMS sketch, we snap s and/or e to the nearest interval end point.

The error from the snapping operation in the leaf-level is at most $O(\varepsilon M_k'')$. By the property of the AMS sketch [17], the approximation error of any AMS sketch in the global Q-AMS sketch is at most $O(\varepsilon F_2(I))$, with at least probability $(1-\delta)$, for an interval I covered by that AMS sketch. Also $F_2(I) \le M_k''$ for any I in the global Q-AMS sketch. Furthermore, there are at most $O(\log \frac{1}{\varepsilon})$ intervals in a canonical cover since the height of the tree in Q-AMS is $\log\lceil \frac{1}{\varepsilon} \rceil$. Hence, the approximation error for any range F_2 query in the global Q-AMS sketch is $O(\varepsilon M_k'' \log \frac{1}{\varepsilon})$ with probability at least $(1-\delta)$, for $\varepsilon, \delta \in (0, 1)$ used in the construction of the Q-AMS sketch. Finally, the size of an AMS sketch is $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ [12, 17]. Thus, we can show that:

Lemma 6 *Given the partition boundaries* $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ *for a sub-domain* $[s_k, e_k]$*, for any* s, e *such that* $s_k \le s < e \le e_k$*, Q-AMS can approximate* $(\mathrm{E}_{\mathcal{W}}[g_s])^2 + (\mathrm{E}_{\mathcal{W}}[g_{s+1}])^2 + \cdots + (\mathrm{E}_{\mathcal{W}}[g_e])^2$ *within an additive error of* $O(\varepsilon M_k'' \log \frac{1}{\varepsilon})$ *with probability* $\ge (1-\delta)$ *using space of* $O(\frac{1}{\varepsilon^3} \log \frac{1}{\delta})$.

Communication cost and partition boundaries. Each aggregate node needs to send $(\frac{1}{\varepsilon} - 1)$ values per sub-domain to all β data nodes, and there are m sub-domains in total. So the communication cost of this step is $O(m\beta/\varepsilon)$. Then, each data node needs to send out m local Q-AMS sketches, one for each sub-domain. The communication cost of this step is $O(\frac{m\beta}{\varepsilon^3} \log \frac{1}{\delta})$. Hence, the total communication is $O(\frac{m\beta}{\varepsilon^3} \log \frac{1}{\delta})$, which caters for the worst-case analysis.

But the above method and analysis depend on the calculation of the partition boundaries $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ for any sub-domain $[s_k, e_k]$, for $k \in [1, m]$. To calculate this exactly we need $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_{e_k}]\}$ at the kth aggregate node, which obviously are not available (unless using $O(n\beta)$ total communication for β data nodes for all sub-domains, which defeats our purpose). Fortunately, given that VS can estimate each $B_k[j]$ with an ε error efficiently, each $\mathrm{E}_{\mathcal{W}}[g_i]$ can be estimated as $(\widehat{B}_k[i] - \widehat{B}_k[i-1])$ By (9).

6. EXPERIMENTS

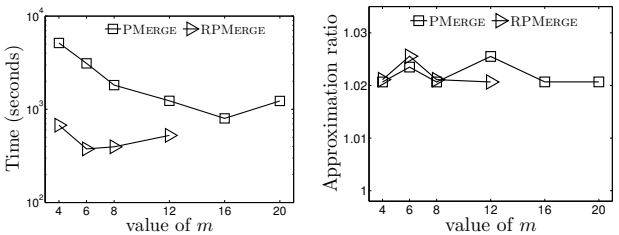
We implemented all methods in Java. We test OptHist, EF-Histogram, Pmerge and RPmerge methods in centralized environment without parallelism, and parallel-Pmerge and parallel-RPmerge methods (with and without synopsis) in distributed and parallel settings. The centralized experiments were executed over a Linux machine running a single Intel i7 3.2GHz cpu, with 6GB of memory and 1TB disk space. We then used MapReduce as the distributed and parallel programming framework and tested all methods in a Hadoop cluster with 17 machines (of the above configuration) running Hadoop 1.0.3. The default HDFS (Hadoop distributed file system) chunk size is 64MB.

Datasets. We executed our experiments using the WorldCup data set and the SAMOS data set. The WorldCup data set is the access logs of 92 days from the 1998 World Cup servers, composed of 1.35 billion records. Each record consists of client id, file type and time of access etc. We choose the client id as the item domain, which has a maximum possible domain size of 2,769,184. We vary the domain size of client ids from 10,000 up to 1,000,000. Records in the entire access log are divided into continuous but disjoint groups, in terms of access time. We generate a discrete frequency distribution pdf for items within each grouping interval and assign the pdf to a tuple in the *tuple model*. For the *value model*, we derive a discrete pdf for each client id based on its frequency distribution in the whole log with respect to 13 distinct requested file types and assign the pdf to the tuple with that client id in the value model. The SAMOS data set is composed of 11.8 million records of various atmospheric measurements from a research vessel and we care about the temperature field, which has a domain size of about 10,000 (by counting two digits after the decimal point of a fraction reading). In a similar way, we form the tuple model and value model data on the SAMOS data.

Setup. The default data set is WorldCup. To accommodate the limited scalability of OptHist, we initially vary the value of n from 10,000 up to 200,000 and test the effects of different parameters. The default values of parameters are $B = 400$ and $n = 100,000$. For RPmerge, the recursion depth is $\ell = 2$. We set $m = 16$ and $m = 6$ as the default values for Pmerge and RPmerge respectively. We then explore the scalability of our methods, up to a domain size of $n = 1,000,000$. The running time of all methods are only linearly dependent on N, number of tuples in a database. Hence, we did not show the effect of N; all reported running time are already *start-to-end wall-clock* time.

In each experiment, unless otherwise specified, we vary the value of one parameter, while using the default values of other parameters. The approximation ratios of our approximate methods were calculated with respect to the optimal B-buckets histogram produced by OptHist [6, 7].

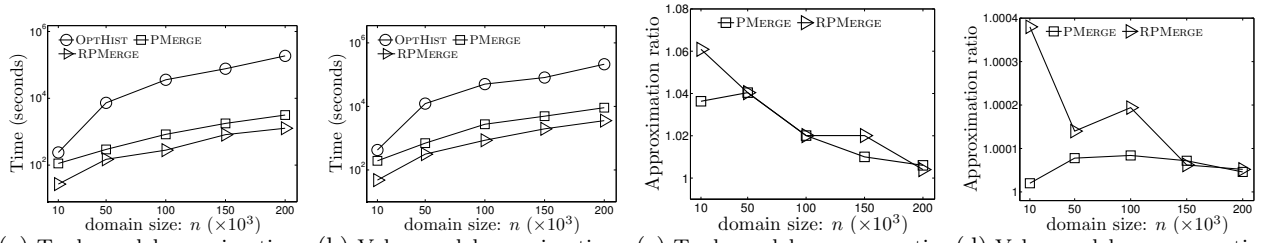
6.1 Centralized environment



(a) m vs running time. (b) m vs approx. ratio.

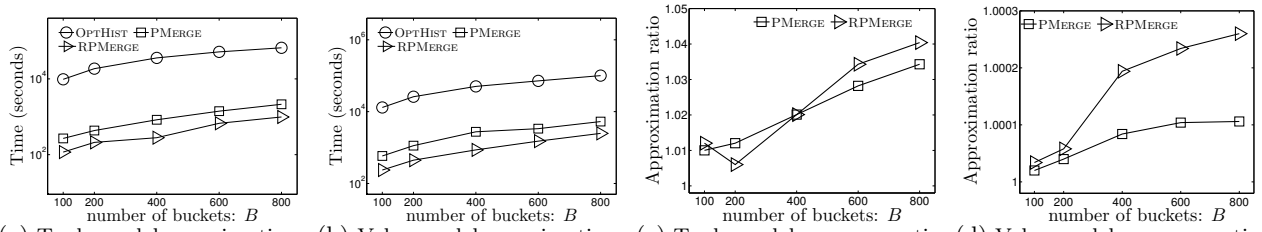
Figure 4: Vary m on the tuple model.

Effect of m. Figure 4 shows the running time and approximation ratio when we vary m from 4 to 20 on the tuple model data sets. Recall that Pmerge will produce m sub-domains, while RPmerge will produce m^ℓ sub-domains, in the partition phase. Hence, RPmerge gives the same number of sub-domains using a (much) smaller m value. For both methods, a larger m value will reduce the size of each



(a) Tuple model: running time. (b) Value model: running time. (c) Tuple model: approx. ratio. (d) Value model: approx. ratio.

Figure 5: Approximation ratio and running time: vary n.



(a) Tuple model: running time. (b) Value model: running time. (c) Tuple model: approx. ratio. (d) Value model: approx. ratio.

Figure 6: Approximation ratio and running time: vary B.

sub-domain, hence, reducing the runtime of the OptHist method on each sub-domain and the overall cost of the partition phase. But a larger m value increases the cost of the merge phase. As a result, we expect to see a sweet point of the overall running time across all m values. Figure 4(a) reflects exactly this trend and the same trend holds on the value model data set as well. They consistently show that $m = 16$ and $m = 6$ provide the best running time for PMerge and RPMerge respectively. Note that this sweet point can be analytically analyzed, by taking derivative of the cost function (partition phase + merge phase) with respect to m.

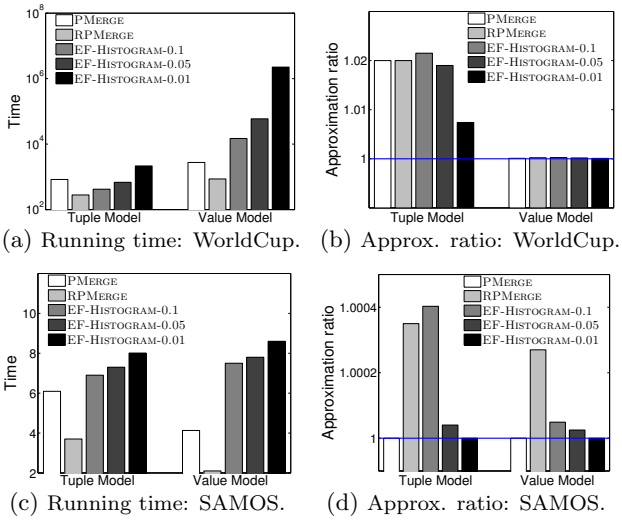
Figure 4(b) shows their approximation ratios on the tuple model data set. The approximation quality of both methods fluctuates slightly with respect to m; but they both produce B-buckets histograms of extremely high quality with approximation ratio very close to 1. The quality is much better than their worst-case theoretical bounds, as indicated by Theorems 1 and 2 respectively.

The results of varying m from the value model are very similarly, and have been omitted for brevity. Also, we have investigated the results of varying the recursive depth ℓ from 1 to 3. They consistently show that $\ell = 2$ achieves a nice balance between running time and approximation quality. For brevity, we ommited the detailed results.

Effect of n. Figure 5 shows the results with respect to n on both value and tuple models. In both models, the running time of OptHist increases quadratically with respect to n. In contrast, both PMerge and RPMerge are much more scalable, and have outperformed OptHist by at least one to two orders of magnitude in all cases. For example, in Figure 5(b), when $n = 100,000$, OptHist took nearly 14 hours while RPMerge took only 861 seconds. RPMerge further improves the running time of PMerge by about 2-3 times and is the most efficient method.

Meanwhile, both PMerge and RPMerge achieve close to 1 approximation ratios across all n values in Figures 5(c) and Figure 5(d). The approximation quality gets better (approaching optimal) as n increases on both models.

Effect of B. We vary the number of buckets from 100 to 800 in Figure 6. Clearly, RPMerge outperforms OptHist by two orders of magnitude in running time in both models, as see in Figures 6(a) and 6(b). Figures 6(c) and 6(d) show the approximation ratios in each model respectively. The approximation ratio of both PMerge and RPMerge slightly increases when B increases on both models. Nevertheless, the quality of both methods are still excellent, remaining very close to the optimal results in all cases.



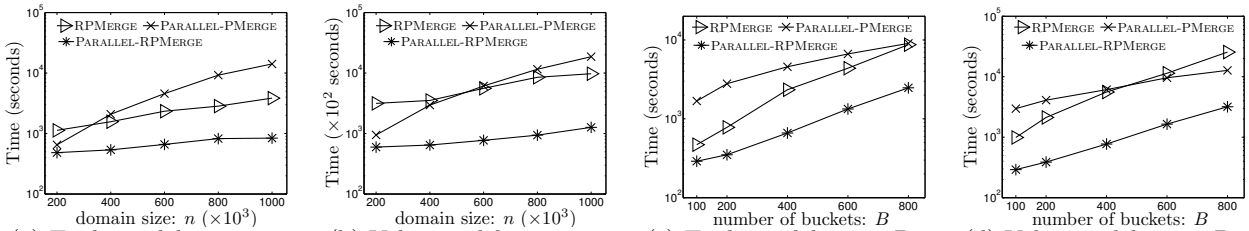
(a) Running time: WorldCup. (b) Approx. ratio: WorldCup.

(c) Running time: SAMOS. (d) Approx. ratio: SAMOS.

Figure 7: Comparison against the baseline method.

Comparison with the baseline. Lastly, we compare the running time and approximation ratios of our methods against the baseline EF-Histogram method (with $\varepsilon = 0.1, \varepsilon = 0.05$ and $\varepsilon = 0.01$ respectively) on two data sets. Our methods used their default parameter values on the WorldCup data set. For the SAMOS data set, we set $n = 10,000$ and $B = 100$. Clearly, small ε values does help improve the approximation quality of EF-Histogram as shown in Figure 7(b) and Figure 7(d). But our methods have provided almost the same approximation quality on both data sets, while offering worst-case bounds in theory as well. Note that EF-Histogram only provides the $(1 + \varepsilon)$ approximation bound with respect to the B-buckets histogram on expected frequencies, but not on the probabilistic histograms.

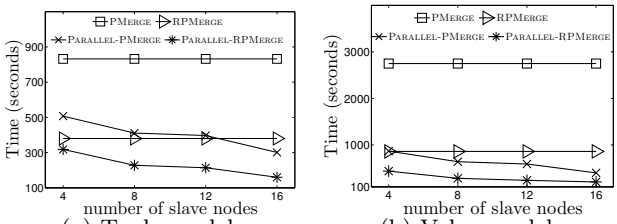
Meanwhile, the running time of EF-Histogram increases significantly (it is actually quadratic to the inverse of ε value,



(a) Tuple model: vary n. (b) Value model: vary n. (c) Tuple model: vary B. (d) Value model: vary B.

Figure 8: Scalability of the parallel approximate methods.

i.e., $1/\varepsilon^2$), especially on the much larger WorldCup data set. In all cases our best centralized method, RPMERGE, has significantly outperformed the EF-Histogram as shown in Figure 7(a) and Figure 7(c). Furthermore, the distributed and parallel fashion of PMERGE and RPMERGE further improves the efficiency of these methods, as shown next.



(a) Tuple model. (b) Value model.

Figure 9: Time: vary number of slave nodes.

6.2 Distributed and parallel setting

Effect of size of the cluster. Figure 9 shows the running time of different methods when we vary the number of slave nodes in the cluster from 4 to 16. For reference, we have included the running time of centralized PMERGE and RPMERGE. We can see a (nearly) linear dependency between the running time and the number of slave nodes for both parallel-PMERGE and parallel-RPMERGE methods. The speed up for both methods is not as much as the increasing factor of the number of slave nodes used. The reason is that Hadoop always includes some extra overhead such as job launching and tasks shuffling and IO cost of intermediate HDFS files, which reduces the overall gain from parallelism.

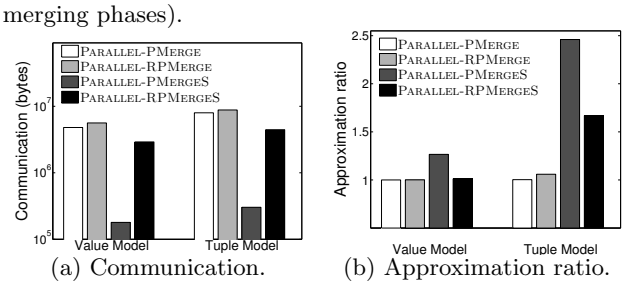
Scalability. Next, we investigate the scalability of RPMERGE (the best centralized method), parallel-PMERGE and parallel-RPMERGE on very large probabilistic data sets. We used all 16 slave nodes in the cluster, and varied either the values of n from 200,000 to 1000,000 when $B = 400$, or the values of B from 100 to 800 when $n = 600,000$. We omit OPTHIST and PMERGE methods in this study, since they are too expensive compared to these methods.

Figures 8(a) and 8(b) show that with recursive merging RPMERGE can even outperform parallel-PMERGE as n increases. But clearly parallel-RPMERGE is the best method and improves the running time of RPMERGE by 8 times on the value model and 4 times on the tuple model when $n = 1000,000$. It becomes an order of magnitude faster than parallel-PMERGE in both models when n increases.

Figures 8(c) and 8(d) show the running time when we vary B and fix $n = 600,000$. Running time of all methods increase with larger B values. This is because large B values increase the computation cost of the merging step, especially for recursive PMERGE. Nevertheless, parallel-RPMERGE significantly outperforms both parallel-PMERGE and RPMERGE in all cases on both models.

6.3 Distributed and parallel synopsis

Lastly, we study the communication saving and approximation quality of parallel-PMERGE and parallel-RPMERGE with synopsis. The default values are $n = 600,000$, $B = 400$ and $\varepsilon = 0.002$ for VS and $\varepsilon = 0.1$ for TS. We have omitted the results for the running time of Parallel-PMERGE and Parallel-RPMERGE with synopsis, since they are very close to that of Parallel-PMERGE and Parallel-RPMERGE respectively (since the running time of all these methods are dominated by solving the DP instances in the partition and merging phases).



(a) Communication. (b) Approximation ratio.

Figure 10: Effects of using synopsis.

Comparing effects of synopsis in both models. Here we use parallel-PMERGES (parallel-RPMERGES) to denote a parallel-PMERGE (parallel-RPMERGE) method with a synopsis in either model. In value model, the synopsis is VS; and in tuple model, the synopsis is TS.

Figure 10(a) shows that parallel-PMERGES outperforms parallel-PMERGE and parallel-RPMERGE by more than an order of magnitude in communication cost for both models. Parallel-RPMERGES has much higher communication cost than parallel-PMERGES since the sampling cost in the partition phase has increased by an order of m using m^2 subdomains (when $\ell = 2$). Nevertheless, it still saves about 2-3 times of communication cost compared to that of parallel-PMERGE and parallel-RPMERGE for both models.

Figure 10(b) shows that parallel-PMERGES and parallel-RPMERGES have excellent approximation quality on the value model (very close to optimal histograms). They give less optimal approximations in the tuple model, since Q-AMS in the TS method has higher variances in its estimated A, B arrays in the tuple model, compared to the estimations on A, B arrays given by VS in the value model.

Remarks. The communication cost of all of our synopsis methods are independent of n, whereas the communication cost of both parallel-PMERGE and parallel-RPMERGE are linearly dependent on n, as shown from our analysis in Section 5. This means the synopsis methods introduce even more savings when domain size increases.

7. RELATED WORK

We have reviewed the most relevant related work in Section 2. That said, extensive efforts were devoted to constructing histograms in deterministic data, motivated by the early work in [14–16,19]. An extensive survey for histograms on deterministic data is in [13]. There are also numerous efforts on modeling, querying, and mining uncertain data; see [1, 3, 21, 24]. A good histogram for large probabilistic data is very useful for many such operations, e.g, finding frequent items, patterns, and itemsets [1, 3, 24, 26].

However, little is known about histograms over probabilistic data till three recent studies [5–7]. Cormode and Garofalakis have extended the bucket-based histogram and the wavelet histogram to probabilistic data by seeking to minimize the expectation of bucket errors over all possible worlds [6, 7]. The details of which can be found in Section 2. Cormode and Deligiannakis then extend the probabilistic histogram definition to allowing bucket with a pdf representation rather than a single constant value [5]. A main limitation of these studies is the lack of scalability, when the domain size of the probabilistic data increases.

Allowing some approximations in histogram construction is also an important subject on deterministic data, e.g., [11, 22, 23] and many others. One possible choice is to run these methods on expected frequencies of all items, and simply use the output as an approximation to our histogram. But the theoretical approximation bound with respect to the deterministic data (in our case, the expected frequencies of all items) does not carry over to probabilistic histogram definition with respect to n random variables (frequency distributions of every item i). To the best of our knowledge, the $(1+\varepsilon)$ approximation from [23] is the best method with theoretical guarantees for histograms over deterministic data (in fact, to the best of our knowledge, other methods are mostly heuristic-based approaches). We did explore this approach as a baseline method in our study.

8. CONCLUSION

This paper designed novel approximation methods for constructing optimal histograms on large probabilistic data. Our approximations run much faster and have much better scalability than the state-of-the-art. The quality of the approximate histograms are almost as good as the optimal histograms in practice. We also introduced novel techniques to extend our methods to distributed and parallel settings, which further improve the scalability. Interesting future work include but not limited to how to extend our study to probabilistic histograms with pdf bucket representatives [5] and how to handle histograms of other error metrics.

9. ACKNOWLEDGMENT

Mingwang Tang and Feifei Li were supported in part by NSF Grants IIS-1251019 and IIS-1200792.

10. REFERENCES

[1] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *SIGKDD*, 2009.
[2] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.
[3] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic frequent itemset mining in uncertain databases. In *SIGKDD*, 2009.
[4] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.
[5] G. Cormode and A. Deligiannakis. Probabilistic histograms for probabilistic data. In *VLDB*, 2009.
[6] G. Cormode and M. Garofalakis. Histograms and wavelets on probabilistic data. In *ICDE*, 2009.
[7] G. Cormode and M. Garofalakis. Histograms and wavelets on probabilistic data. *IEEE TKDE*, 22(8):1142–1157, 2010.
[8] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.
[9] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
[10] X. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, 2007.
[11] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *VLDB*, 1997.
[12] M. G. Graham Cormode and D. Sacharidis. Fast approximate wavelet tracking on streams. In *EDBT*, 2006.
[13] Y. E. Ioannidis. The history of histograms. In *VLDB*, 2003.
[14] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, 1995.
[15] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 1998.
[16] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.
[17] Y. M. Noga Alon and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, 1996.
[18] J. Pei, M. Hua, Y. Tao, and X. Lin. Query answering techniques on uncertain and probabilistic data: tutorial summary. In *SIGMOD*, 2008.
[19] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, 1996.
[20] A. D. Sarma, O. Benjelloun, A. Y. Halevy, S. U. Nabar, and J. Widom. Representing uncertain data: models, properties, and algorithms. *VLDBJ*, 18(5):989–1019, 2009.
[21] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. 2011.
[22] K. S. Sudipto Guha and J. Woo. Rehist: Relative error histogram construction algorithms. In *VLDB*, 2004.
[23] N. K. Sudipto Guha and K. Shim. Approximation and streaming algorithms for histogram construction problems. *TODS*, 31(1):396–438, 2006.
[24] L. Sun, R. Cheng, D. W. Cheung, and J. Cheng. Mining uncertain data with probabilistic guarantees. In *SIGKDD*, 2010.
[25] M. Tang and F. Li. Scalable histograms on large probabilistic data. In *Technical report, http://www.cs.utah.edu/∼lifeifei/papers/ph.pdf*, 2014.
[26] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *SIGMOD*, 2008.

Appendix A: The Weighted Histogram

Fast computation of bucket Error. We can show that in the weighted case the $\min_{\widehat{b}}(s, e, \widehat{b})$ is achieved by setting $\widehat{b} = \frac{\sum_{k=s}^{e} w_k f_k}{\sum_{k=s}^{e} w_k}$ and the corresponding bucket error for the bucket b is as follows: $SSE(b, \widehat{b}) = \sum_{j=s}^{e} w_j (f_j^2 - \widehat{b}^2)$. The prefix sum arrays need to be precomputed are:

$$P[e] = \sum_{i=1}^{e} w_j f_j, \quad PP[e] = \sum_{i=1}^{e} w_j f_j^2, \quad W[e] = \sum_{i=1}^{e} w_j.$$

Given these arrays, $\min_{\widehat{b}}(s, e, \widehat{b})$ is computed as:

$$PP[e] - PP[s] - \frac{(P[e] - P[s])^2}{W[e] - W[s]}.$$