Optimal Sampling Algorithms for Frequency Estimation in Distributed Data

Zengfeng Huang Ke Yi Yunhao Liu Guihai Chen

Hong Kong University of Science and Technology
{huangzf, yike, liu}@cse.ust.hk

Shanghai Jiaotong University
gchen@nju.edu.cn

***Abstract*—Consider a distributed system with n nodes where each node holds a multiset of items. In this paper, we design sampling algorithms that allow us to estimate the global frequency of any item with a standard deviation of εN, where N denotes the total cardinality of all these multisets. Our algorithms have a communication cost of $O(n + \sqrt{n}/\varepsilon)$, which is never worse than the $O(n + 1/\varepsilon^2)$ cost of uniform sampling, and could be much better when $n \ll 1/\varepsilon^2$. In addition, we prove that one version of our algorithm is *instance-optimal* in a fairly general sampling framework. We also design algorithms that achieve optimality on the bit level, by combining Bloom filters of various granularities. Finally, we present some simulation results comparing our algorithms with previous techniques. Other than the performance improvement, our algorithms are also much simpler and easily implementable in a large-scale distributed system.**

I. Introduction

Consider a distributed system with n nodes where each node holds a list of (item, frequency) pairs, recording the local frequency of these items. In this paper, we study the problem of estimating the global frequencies of the items where an item's global frequency is the sum of all the local frequencies at all the nodes, with minimum communication. This problem is motivated by many applications in distributed databases, network monitoring, sensor networks, data centers, cloud computing, etc. For example, estimating the frequencies of queried keywords is a routine task for search engines, while the query logs have to be stored in a distributed manner due to their sheer scale. The recently developed MapReduce framework [8] has provided a highly efficient and reliable programming environment for processing massive distributed data sets. As another example, in DDoS attacks the attacker tries to send a lot of traffic to the same victim via many different routes, so any individual router may not see a large number of packets destined to the victim. Thus in order to detect DDoS attacks we will have to estimate the global frequency of destination IP addresses. Other examples include estimating the popularity of files in peer-to-peer file-sharing networks, reporting the occurrences of different species of birds in a sensor network, and so on.

A. Problem definition

We assume that the items are drawn from a bounded universe $[u] = \{1, \dots, u\}$. There are n distributed nodes in the system; we denote the *local count* of item i at node j by $x_{i,j}$, and the *global count* of item i is $y_i = \sum_j x_{i,j}$. The total count of all items is denoted $N = \sum_i y_i$. There is a coordinator C whose job is to estimate y_i for all $i \in [u]$ by communicating with the nodes with minimum cost. Here and further the word "cost" will always refer to "communication cost".

Since computing all the y_i's exactly incurs high costs and is often unnecessary, we will allow an absolute error of at most εN for some small $\varepsilon > 0$. When probabilistic algorithms are concerned, this should be achieved with at least constant probability. This error definition has been used in most works on this problem [6, 7, 12, 13, 19]. Under such an error definition, we can zero out all the frequencies less than εN, so that the output size is bounded by $1/\varepsilon$. On the other hand, if we use a relative ε-error, then we are forced to make very accurate estimations for low-frequency items, which is expensive yet unnecessary. Note that there are some proposals of a (p, ε)-error [2, 11] that guarantees a relative ε-error only for frequencies at least pN, which we briefly discuss in Section VII.

B. Preliminaries and previous results

There is a simple deterministic algorithm with cost $O(n/\varepsilon)$. The idea is to ask each node to send in all its items with local counts greater than $\varepsilon N/n$. Thus, for any item, each node contributes an error at most $\varepsilon N/n$, totaling εN from all the n nodes. This simple algorithm has been used in some previous work [5], and is conjectured to be optimal for deterministic algorithms, although there has not been a proof.

With randomization there is potential to do better. To start with, it is well known [18] that uniformly sampling each item with probability $p = 1/\varepsilon^2 N$ suffices to estimate the count of any item within an error of εN with constant probability. To do the sampling, we need to compute N and then broadcast p to all nodes, which require $O(n)$ communication. The total (expected) cost of the sampling is thus $O(n + pN) = O(n + 1/\varepsilon^2)$. So uniform sampling beats the deterministic algorithm when $n > 1/\varepsilon$. But how about the case $n < 1/\varepsilon$, which is more likely in real applications? Bear in mind that the ε-error as defined is an absolute error of εN, where N is the total count of *all* items, so typical values of ε range from 0.0001 to 0.01 (see e.g. [6]), while n in real systems is usually no more than a few hundred.

Zhao et al. [21] defined a general sampling framework for the frequency estimation problem. Let $g : \mathbb{N} \to [0, 1]$ be a *sampling function*. If an item has local count x at a node, then the node with probability $g(x)$ samples this item and sends the item together with its local count x to the coordinator. In addition to the local count x, we also allow the function $g(x)$ to depend on N, n, ε. Set $Y_{i,j} = x_{i,j}$ if the coordinator receives the (item, count) pair $(i, x_{i,j})$ from node j, and $Y_{i,j} = 0$ otherwise. Then the coordinator can estimate y_i using (define $\frac{0}{0} = 0$)

$$Y_i = \frac{Y_{i,1}}{g(Y_{i,1})} + \cdots + \frac{Y_{i,n}}{g(Y_{i,n})}, \tag{1}$$

which was shown [21] to be an unbiased estimator with variance

$$\mathrm{Var}[Y_i] = \sum_{j=1}^{n} \mathrm{Var}\left[\frac{Y_i}{g(x_{i,j})}\right] = \sum_{j=1}^{n} \frac{x_{i,j}^2(1 - g(x_{i,j}))}{g(x_{i,j})}. \tag{2}$$

This framework is more general and should intuitively do better than uniform sampling. But as pointed out in [21], central to this framework is the choice of the sampling function g. To be able to estimate y_i with error at most εN with a constant probability, we need to choose a g such that $\mathrm{Var}[Y_i] = O((\varepsilon N)^2)$. From (2) it is clear that $\mathrm{Var}[Y_i]$ gets smaller as g gets larger. On the other hand, the expected total number of (item, count) pairs sent to the coordinator is $\sum_{i,j} g(x_{i,j})$, so we would want to choose the smallest g such that $\mathrm{Var}[Y_i] = O((\varepsilon N)^2)$.

Zhao et al. [21] proposed to use $g(x) = x/(x+d)$ for some fixed d. When using such a g, $\mathrm{Var}[Y_i]$ simplifies to dy_i. Since y_i can be as large as $\Theta(N)$, we will have to set $d = \Theta(\varepsilon^2 N)$ in order to guarantee $\mathrm{Var}[Y_i] = O((\varepsilon N)^2)$ for any i. Thus, the communication cost of their algorithm is $\sum_{i,j} g(x_{i,j}) = \sum_{i,j} \frac{x_{i,j}}{d+x_{i,j}}$. In the worst case (when all the $x_{i,j}$'s are no more than d), this is at least $\frac{1}{2}\sum_{i,j} \frac{x_{i,j}}{d} = \Theta(N/d) = \Theta(1/\varepsilon^2)$. We also need to compute N and broadcast d to all nodes, so the total cost is $\Theta(n + 1/\varepsilon^2)$, which is the same as that of uniform sampling. [21] also proposed some other complicated heuristics, but they do not improve the $\Theta(n + 1/\varepsilon^2)$ worst-cast cost. Therefore, although Zhao et al. [21] proposed a nice general sampling framework, they did not demonstrate whether this framework yields a solution that is asymptotically better than uniform sampling.

C. *Our results*

In this paper we answer the above question in the affirmative, and in a very strong sense. Specifically, we obtain the following results.

We first show in Section III that the linear sampling function[1] $g_1(x) = x\sqrt{n}/\varepsilon N$ achieves $\mathrm{Var}[Y_i] \le O((\varepsilon N)^2)$, which, by Chebyshev's inequality, allows us to estimate y_i within an error of εN with constant probability. The communication cost is $\sum_{i,j} g(x_{i,j}) = O(\sqrt{n}/\varepsilon)$ under any input[2]. This is clearly better than the $O(n/\varepsilon)$ deterministic bound. It is also much better than the $O(1/\varepsilon^2)$ uniform sampling cost when $n \ll 1/\varepsilon^2$. In the (rare) case $n > 1/\varepsilon^2$, uniform sampling still performs better.

Next, we prove an $\Omega(\min\{\sqrt{n}/\varepsilon, 1/\varepsilon^2\})$ lower bound on the worst-case cost for all *valid* sampling functions. A sampling function is *valid* if it achieves $\mathrm{Var}[Y_i] \le O((\varepsilon N)^2)$ for all i under any input. This means that sampling with g_1 and uniform sampling are respectively optimal in the cases $n < 1/\varepsilon^2$ and $n > 1/\varepsilon^2$.

Although we have found optimal sampling functions for all values of n and ε, these are actually not the main results of this paper. We observe that on some inputs, it is possible to further reduce the communication cost. Consider an extreme case where all the $x_{i,j}$'s are either 0 or 1. If we use g_1, the cost is $\Theta(\sqrt{n}/\varepsilon)$. (In fact, due to the "linear" feature of g_1, its cost $\sum_{i,j} g_1(x_{i,j})$ is almost always $\Theta(\sqrt{n}/\varepsilon)$.) In this case, g_1 samples each $x_{i,j} = 1$ with probability $\sqrt{n}/\varepsilon N$. However, we observe from (2) that when the $x_{i,j}$'s are all very small, we can afford to use a smaller sampling rate while still keeping $\mathrm{Var}[Y_i]$ small. Indeed, in this case we can sample each $x_{i,j} = 1$ with probability $n/(\varepsilon N)^2$ while still having $\mathrm{Var}[Y_i] = O((\varepsilon N)^2)$. When using such a sampling rate, the total cost reduces to $\Theta(Nn/(\varepsilon N)^2) = \Theta(n/\varepsilon^2 N)$,

[1]The function is actually $g_1(x) = \min\{x\sqrt{n}/\varepsilon N, 1\}$ as any g cannot exceed 1. We omit the "min" here and further for brevity.

[2]Henceforth we omit the $O(n)$ term for computing N and broadcasting g to all nodes when considering the algorithms' costs, because 1) it makes the bounds cleaner; 2) this cost is common to all algorithms in this sampling framework; and 3) this cost is due to computing N, which can be easily shown to be unavoidable if N is unknown.

which, interestingly, approaches 0 as $N \to \infty$, and can be much lower than both the $\Theta(\sqrt{n}/\varepsilon)$ cost of g_1 and the $\Theta(1/\varepsilon^2)$ cost of uniform sampling. Although practical cases are not as extreme, it is very common to have a lot of small $x_{i,j}$'s due to the heavy tail property of many real-world distributions. Of course, in view of the above lower bound, it is not possible to beat (the better of) g_1 and uniform sampling on the worst input, but can we do something better for these typical, not-so-worst cases?

The answer is yes. In Section IV we show that the sampling function

$$g_2(x) = \min\{x^2 n/(\varepsilon N)^2, x/\varepsilon^2 N\}$$

also achieves $\mathrm{Var}[Y_i] \le O((\varepsilon N)^2)$ with the optimal worst-case cost of $O(\min\{\sqrt{n}/\varepsilon, 1/\varepsilon^2\})$. So in the worst case, it performs the same as g_1 (for $n < 1/\varepsilon^2$) and uniform sampling (for $n > 1/\varepsilon^2$). To analytically establish the superiority of g_2, we prove that it is *instance-optimal* [9], i.e., for every given input $I : \{x_{i,j}\}$, g_2 has the optimal cost (up to a constant factor) among all the valid sampling functions on that input. More precisely, let $opt(I) = \sum_{i,j} g_2(x_{i,j})$ be the cost of sampling with g_2 on input I, we show that any valid sampling function must have cost $\Omega(opt(I))$ on I, which essentially means that g_2 is the best sampling function on every single input. This is a much stronger optimality notion than the traditional worst-case optimality.

So far we have treated an (item, count) pair as a communication unit. If one desires a more precise analysis, such a pair actually consumes $O(\log u + \log N)$ bits. To further reduce the communication cost, we design techniques that are more careful about the bits they send. We show in Section V how to reduce the communication cost on the bit level by using multiple Bloom filters [1, 16] at different levels of granularity. Although g_1 is not as good as g_2 in terms of sampling, it is particularly amenable to Bloom filters. It turns out that we can remove the extra $O(\log u + \log N)$ factor completely when using g_1, i.e., we obtain an algorithm communicating $O(\sqrt{n}/\varepsilon)$ bits. The instance-optimal sampling function g_2 is more difficult to directly plug into Bloom filters, but then we use an interesting combination of g_1 and g_2 to achieve a cost of $O\left(opt(I)\log^2\left(\frac{\sqrt{n}}{\varepsilon \cdot opt(I)}\right)\right)$ bits.

Finally, we comment that all our algorithms are very simple, and can be easily implemented in a large-scale distributed system. In particular, they can be easily accomplished in the MapReduce framework. Thus, we would claim that our algorithms are both theoretically interesting and practically useful.

II. Related work

Besides the work of Zhao et al. [21] which is the closest work to ours, a number of related problems have been studied by the database and distributed computing communities.

The *heavy hitter* problem [6, 13] has been well studied in the centralized case. The goal here is to report all items with frequency exceeding ϕN for some user specified ϕ, not report items with frequency below $(\phi - \varepsilon)N$, while we do not care frequencies in between. Our algorithms clearly solve this problem in the distributed setting. Zhao et al. [20] studied the distributed heavy hitter problem while using a relative λ-error ($0 < \lambda < 1$): the heavy hitters' global counts are more than τ while the non-heavy hitters' counts should be less than $\lambda\tau$, for some threshold τ. The communication cost of their algorithm is $O(nu\lambda^2)$, so it only applies to situations where there is a very small universe. Furthermore, note that $O(nu)$ is a trivial upper bound (each node sending all items would cost this much), so this algorithm beats the naive solution only when there is a very large gap between the heavy and non-heavy hitters. Meanwhile, they also showed a matching $\tilde{\Omega}(ku\lambda^2)$ lower bound[3] This apparent hardness of the problem actually stems from distinguishing between a global count of 1 and $1/\lambda$, which corresponds to using a very small τ. If $\tau = \phi n$ as mostly used in the literature [6], this problem can be in fact solved efficiently as shown in this paper.

The distributed *top-k* problem [4, 14, 15] is another related one, where the goal is to find the top-k most frequent items. Cao and Wang [4] designed an algorithm that solves this problem exactly, but it could ship all the data to the coordinator on some inputs. They also proved that their algorithm is instance-optimal, but this only holds for inputs following a certain distribution and the optimality ratio is as large as $O(n^2)$. Note that our instance-optimality ratio is a constant and it holds for any input. The apparent difficulty of their approach stems from situations where the k-th frequent item is very close to the $(k+1)$-st one, and they want to detect this exactly. Patt-Shamir and Shafrir [15] considered a more solvable version of the problem where they allow a relative ε-error when separating the top-k list from the rest. The communication cost of their algorithm is $\tilde{O}(1/p^*\varepsilon^2)$ where p^* is the frequency of the k-th frequent item divided by N. Their algorithm uses multiple rounds of uniform sampling to estimate the frequencies. If we use our algorithms in place of uniform sampling, the cost improves to $\tilde{O}(\sqrt{n}/p^*\varepsilon)$ when $n \ll 1/\varepsilon^2$. Using our instance-optimal algorithm will result in a larger

[3]The $\tilde{}$ notation suppresses all polylog factors in $u, N, n, 1/\varepsilon$.

improvement for certain inputs, although it is hard to analytically quantify, since instance optimality has to be stated for a specific problem and in a clearly defined framework. Michel et al. [14] generalized and improved the algorithm of [4], but with no analytical results.

The distributed approximate *quantile* problem has also been well studied, where the goal is to return a set of items whose ranks are between $(\phi-\varepsilon)N$ and $(\phi+\varepsilon)N$ for all $0<\phi<1$. It is known [6] that the frequency estimation problem reduces to the quantile problem, but the best algorithms for the latter incur $\tilde{O}(n/\varepsilon)$ costs [10, 17], at least a factor $\tilde{O}(\sqrt{n})$ worse than our bounds.

Finally, there have been a lot of interests in the problem of *continuously tracking* the heavy hitters, quantiles, and top-k items in distributed data [3, 5, 19]. The tracking problem is more general as it requires solving the respective problems at all times continuously, rather than a one-shot computation. However, all these cited works studied only deterministic algorithms; in fact, the deterministic complexity for tracking the heavy hitters and quantiles has been settled at $\tilde{\Theta}(n/\varepsilon)$ in [19]. We believe that the techniques developed in this paper could lead to probabilistic schemes solving these problems with cost $\tilde{O}(\sqrt{n}/\varepsilon)$.

III. A Worst-Case Optimal Sampling Function

In this section, we show that the sampling function $g_1(x)=x\sqrt{n}/\varepsilon N$ is worst-case optimal. In this section and Section IV, we measure the communication cost as the expected total number of (item, count) pairs sampled and sent to the coordinator, i.e., $\sum_{i,j} g(x_{i,j})$ for a given g. In Section V we will conduct a more precise analysis measuring the cost in terms of the bits communicated.

Theorem 3.1: The sampling function $g_1(x) = x\sqrt{n}/\varepsilon N$ has a cost of $O(\sqrt{n}/\varepsilon)$ and achieves $\mathrm{Var}[Y_i]=\frac{1}{4}(\varepsilon N)^2$ for all i.

Proof: The analysis of the cost is trivial: $\sum_{i,j} g_1(x_{i,j}) \le \sum_{i,j} x_{i,j}\sqrt{n}/\varepsilon N = N\cdot\sqrt{n}/\varepsilon N = \sqrt{n}/\varepsilon$.

Now we consider the variance of Y_i. Since we sample an item with probability one (i.e., zero variance) when the local count $x_{i,j}>\varepsilon N/\sqrt{n}$, it is sufficient to consider the worst case when all $x_{i,j}\le \varepsilon N/\sqrt{n}$. By (2), we have

$$\begin{aligned}\mathrm{Var}[Y_i] &= \sum_{j=1}^{n}\frac{x_{i,j}^2(1-x_{i,j}\sqrt{n}/\varepsilon N)}{x_{i,j}\sqrt{n}/\varepsilon N}\\ &= \frac{\varepsilon N}{\sqrt{n}}\sum_{j=1}^{n}x_{i,j}-\sum_{j=1}^{n}x_{i,j}^2\\ &\le \frac{\varepsilon N}{\sqrt{n}}y_i-\frac{1}{n}y_i^2 \qquad \text{(Cauchy-Schwartz)} \quad (3)\\ &= -\left(\frac{y_i}{\sqrt{n}}-\frac{\varepsilon N}{2}\right)^2+\frac{(\varepsilon N)^2}{4}\le\frac{1}{4}(\varepsilon N)^2.\end{aligned}$$

■

Next we establish the worst-case optimality of g_1 when $n<1/\varepsilon^2$. For the (unrealistic) case $n>1/\varepsilon^2$, uniform sampling turns out to be optimal already. Recall that a sampling function g is *valid* if there exists some constant c such that on any input, g achieves $\mathrm{Var}[Y_i]\le c(\varepsilon N)^2$ for all i.

Theorem 3.2: Any valid sampling function has cost $\Omega(\min\{\sqrt{n}/\varepsilon, 1/\varepsilon^2\})$ on some input.

Proof: Let g be any valid sampling function. We will consider the following two cases, respectively.

1) $n<1/\varepsilon^2$: In this case, we consider an input where $x_{i,j}=\varepsilon N/\sqrt{n}$ for all i,j. Thus each item has $y_i = nx_{i,j}=\varepsilon\sqrt{n}\cdot N$ copies over all n nodes, and there are $N/x_{i,j}=\sqrt{n}/\varepsilon$ such $x_{i,j}$'s.

From (2) we have

$$\mathrm{Var}[Y_i]=\sum_{j=1}^{n}\frac{(\varepsilon N)^2/n\cdot(1-g(x_{i,j}))}{g(x_{i,j})}.$$

Since we require $\mathrm{Var}[Y_i]\le c(\varepsilon N)^2$ and all $x_{i,j}$ are equal, we have $\frac{1-g(x_{i,j})}{g(x_{i,j})}\le c$, or $g(x_{i,j})\ge\frac{1}{1+c}$.

The cost of g is thus

$$\sum_{i,j} g(x_{i,j})=\sqrt{n}/\varepsilon\cdot\frac{1}{1+c}=\Omega(\sqrt{n}/\varepsilon).$$

2) $n>1/\varepsilon^2$: In this case, we consider an input where there is only one item in the universe $u=1$, and it exists only at $1/\varepsilon^2$ nodes with $x_{1,j}=\varepsilon^2 N$ at each of these nodes; the other nodes are empty.

From (2) we have

$$\mathrm{Var}[Y_1]=(\varepsilon N)^2\frac{1-g(\varepsilon^2 N)}{g(\varepsilon^2 N)}.$$

By the requirement that $\mathrm{Var}[Y_1]\le c(\varepsilon N)^2$, we have

$$g(\varepsilon^2 N)\ge\frac{1}{1+c}.$$

The cost of g is thus $1/\varepsilon^2\cdot g(\varepsilon^2 N)=\Omega(1/\varepsilon^2)$. ■

IV. An Instance-Optimal Sampling Function

In this section, we first show that the sampling function

$$g_2(x)=\min\{x^2 n/(\varepsilon N)^2, x/\varepsilon^2 N\}$$

also achieves $\mathrm{Var}[Y_i]=O((\varepsilon N)^2)$. In terms of cost, since $g_2(x)\le g_1^2(x)$, the cost of g_2 is always no more than that of g_1; also since $g_2(x)\le x/\varepsilon^2 N$, its cost is at most $O(1/\varepsilon^2)$. Thus it has the optimal worst-case cost of $O(\min\{\sqrt{n}/\varepsilon, 1/\varepsilon^2\})$. To analytically establish its superiority, we later prove that it is instance-optimal, i.e., for any input I, its cost is optimal among all valid sampling functions for I.

Theorem 4.1: The sampling function $g_2(x)$ achieves $\mathrm{Var}[Y_i] \le O((\varepsilon N)^2)$ for all i.

Proof: Similar to the proof of Theorem 3.1, we can assume that $g_2(x_{i,j}) < 1$ for all j; otherwise its contribution to $\mathrm{Var}[Y_i]$ is zero. From (2), we have

$$\begin{aligned}\mathrm{Var}[Y_i] &= \sum_{j=1}^{n} \frac{x_{i,j}^2(1-g_2(x_{i,j}))}{g_2(x_{i,j})} \le \sum_{j=1}^{n} \frac{x_{i,j}^2}{g_2(x_{i,j})} \\ &= \sum_{j=1}^{n} x_{i,j}^2 \max\left\{\frac{(\varepsilon N)^2}{n x_{i,j}^2}, \frac{\varepsilon^2 N}{x_{i,j}}\right\} \\ &\le \sum_{j=1}^{n}\left(x_{i,j}^2 \frac{(\varepsilon N)^2}{n x_{i,j}^2} + \sum_{j=1}^{n} x_{i,j}^2 \frac{\varepsilon^2 N}{x_{i,j}}\right) \\ &= O((\varepsilon N)^2).\end{aligned}$$

■

To prove that g_2 is instance-optimal, for any input $I : \{x_{i,j}\}$ we write $opt(I) = \sum_{i,j} g_2(x_{i,j})$ which is the cost of g_2. We then show that any valid sampling function on I must have cost $\Omega(opt(I))$.

Theorem 4.2: On input $I : \{x_{i,j}\}$, any valid sampling function must have cost $\Omega(opt(I))$.

Proof: Let g be any valid sampling function, and we will show that $\sum_{i,j} g(x_{i,j}) = \Omega(opt(I))$. We will prove it by contradiction: If $g(x_{i,j}) < \frac{1}{2} g_2(x_{i,j})$ for some $x_{i,j}$, we show that it is possible to construct another input I' (with the same N, n, ε so that g stays the same) on which g fails to achieve $\mathrm{Var}[Y_i] \le c(\varepsilon N)^2$. In the proof we will use $c = 1$ for simplicity; the same proof works for any other constant c by properly adjusting the parameters.

We consider the following two cases:

1) $n < 1/\varepsilon^2$: If $x_{i,j} \le \varepsilon N/\sqrt{n}$, then we construct I' by setting $x'_{i,j} = x_{i,j}$ for all j. Now the global count of item i is $y_i = n x_{i,j} = \varepsilon\sqrt{n} \cdot N \le N$. We set the other $x'_{i,j}$ so that $\sum_{i,j} x'_{i,j} = N$. Thus the variance of Y_i is

$$\mathrm{Var}[Y_i] = n\left(\frac{x_{i,j}^2}{g(x_{i,j})} - x_{i,j}^2\right).$$

Note that when $n < 1/\varepsilon^2$ and $x_{i,j} \le \varepsilon N/\sqrt{n}$, we have $g_2(x_{i,j}) = x_{i,j}^2 n/(\varepsilon N)^2$, so by the assumption that $g(x_{i,j}) < \frac{1}{2} g_2(x_{i,j})$,

$$\mathrm{Var}[Y_i] > n\left(\frac{2(\varepsilon N)^2}{n} - \left(\frac{\varepsilon N}{\sqrt{n}}\right)^2\right) = (\varepsilon N)^2.$$

If $x_{i,j} > \varepsilon N/\sqrt{n} > \varepsilon^2 N$, we construct I' by setting $x'_{i,j} = x_{i,j}$ for $1 \le j \le m$, where $m = \min\{N/x_{i,j}, n\}$. One can check that $m x_{i,j}^2 > (\varepsilon N)^2$ always holds. So

$$\mathrm{Var}[Y_i] = m x_{i,j}^2 \left(\frac{1}{g(x_{i,j})} - 1\right) > (\varepsilon N)^2 \left(\frac{1}{g(x_{i,j})} - 1\right).$$

When $n < 1/\varepsilon^2$ and $x_{i,j} > \varepsilon^2 N$, $g_2(x_{i,j}) = 1$. So by the assumption, we have $g(x_{i,j}) < 1/2$. Thus

$$\mathrm{Var}[Y_i] > (\varepsilon N)^2.$$

2) $n > 1/\varepsilon^2$: If $x_{i,j} \le N/n < \varepsilon N/\sqrt{n}$, we set $x'_{i,j} = x_{i,j}$ for all j. By the assumption, we have $g(x_{i,j}) < \frac{n x_{i,j}^2}{2(\varepsilon N)^2}$, then we have

$$\mathrm{Var}[Y_i] = n\left(\frac{x_{i,j}^2}{g(x_{i,j})} - x_{i,j}^2\right) > n\left(\frac{2(\varepsilon N)^2}{n} - x_{i,j}^2\right) > (\varepsilon N)^2.$$

If $x_{i,j} > N/n$, then we set $x'_{i,j} = x_{i,j}$ for $1 \le j \le N/x_{i,j}$. In this case, $x_{i,j}$ may be either greater than or less than $\varepsilon^2 N$. If it is less than $\varepsilon^2 N$, then $g_2(x_{i,j}) = x_{i,j}/\varepsilon^2 N$, and

$$\mathrm{Var}[Y_i] = \frac{N}{x_{i,j}}\left(\frac{x_{i,j}^2}{g(x_{i,j})} - x_{i,j}^2\right) > \frac{N x_{i,j}}{g(x_{i,j})} - (\varepsilon N)^2 > (\varepsilon N)^2.$$

If $x_{i,j}$ is greater than $\varepsilon^2 N$, we have $g_2(x_{i,j}) = 1$ and hence $g(x_{i,j}) < 1/2$, so

$$\mathrm{Var}[Y_i] = N x_{i,j}\left(\frac{1}{g(x_{i,j})} - 1\right) > (\varepsilon N)^2.$$

Summarizing all cases, g cannot be a valid function if $g(x_{i,j}) < \frac{1}{2} g_2(x_{i,j})$. This holds for all i, j, so $\sum_{i,j} g(x_{i,j}) = \Omega(opt(I))$. ■

V. Reducing Communication by Bloom Filters

In this section we will conduct a more precise analysis of the communication cost in terms of the number of bits transmitted. If the nodes directly send a sampled (item, count) pair to the coordinator, the cost will be $O(\log u + \log N)$ bits per pair. Below we show how to reduce this cost by encoding the sampled items into Bloom filters. Recall that a Bloom filter is a space-efficient encoding scheme that compactly stores a set of items S. The particularly interesting feature is that it uses $O(1)$ bits per item, regardless of the length of the item. A Bloom filter does not have false negatives, but may have a constant false positive probability q for any queried item. More precisely, if the queried item is in S, the answer is always “yes”; if it is not in S, then with probability q it returns “yes” and with probability $1-q$ returns “no”. The false positive probability q can be made arbitrarily small by using $O(\log(1/q))$ bits per item. We omit the details of Bloom filters (see e.g. [1] for general information and [16] for the current state of the Bloom filter), but only point out that the false positive probability q can be computed exactly from $|S|$ and the size of the Bloom filter. These two numbers only require $O(\log|S| + \log\log(1/q))$ bits, so transmitting them together with the Bloom filter does not affect the $O(\log(1/q))$-bit cost per item.

Sampling with g_1, the easy case.: Although g_1 has a higher sampling rate than g_2, its linear feature (when $x \le \varepsilon N/\sqrt{n}$) does have an advantage when it comes to saving bits: $\frac{Y_{i,j}}{g_1(Y_{i,j})}$ is either 0 or $\varepsilon N/\sqrt{n}$ in the estimator (1). This is independent of $x_{i,j}$, which means that for any sampled (item, count) pair, the nodes do not actually need to send the count! Thus, the set of (item, count) pairs a node sends to the coordinator becomes just a set of items, which can be encoded in a Bloom filter. Suppose for now that $x_{i,j} \le \varepsilon N/\sqrt{n}$ for all i, j. In this case, each node j simply samples item i with probability $g_1(x_{i,j})$, then encodes the sampled items into a Bloom filter and sends it to the coordinator. For any $i \in [u]$, suppose among the n Bloom filters that the coordinator has received, Z_i of them asserts its existence, then we can estimate y_i as

$$Y_i = \frac{\varepsilon N}{\sqrt{n}} \cdot \frac{Z_i - nq}{1-q}. \tag{4}$$

We show that (4) is an unbiased estimator and has a small variance. We also note that for the analysis to go through, the nodes need to use independent random hash functions in their Bloom filters.

Lemma 5.1: $\mathbf{E}[Y_i] = y_i; \mathrm{Var}[Y_i] \le \dfrac{(\varepsilon N)^2}{4(1-q)^2}$.

Proof: We define $Z_{i,j}$ to be the indicator random variable set to 1 if the Bloom Filter from node j asserts that it contains the item i, and 0 otherwise. It is easy to see that $\Pr[Z_{i,j} = 1] = g_1(x_{i,j}) + (1 - g_1(x_{i,j}))q$, and thus $\mathbf{E}[Z_{i,j}] = g_1(x_{i,j}) + (1-g_1(x_{i,j}))q$. Then we have

$$\begin{aligned}
\mathbf{E}[Y_i] &= \frac{\varepsilon N}{\sqrt{n}} \cdot \frac{\mathbf{E}[Z_i] - nq}{1-q} \\
&= \frac{\varepsilon N}{\sqrt{n}} \cdot \frac{\sum_{j=1}^n \mathbf{E}[Z_{i,j}] - nq}{1-q} \\
&= \frac{\varepsilon N}{\sqrt{n}} \cdot \frac{(1-q)\sum_{j=1}^n g_1(x_{i,j}) + nq - nq}{1-q} \\
&= \frac{\varepsilon N}{\sqrt{n}} \sum_{j=1}^n g_1(x_{i,j}) = y_i.
\end{aligned}$$

The variance of the estimator is

$$\begin{aligned}
\mathrm{Var}[Y_i] &= \frac{(\varepsilon N)^2}{n(1-q)^2}\mathrm{Var}[Z_{i,j}] \\
&= \frac{(\varepsilon N)^2}{n(1-q)^2} \sum_{j=1}^n \mathrm{Var}[Z_{i,j}] \\
&= \frac{(\varepsilon N)^2}{n(1-q)^2} \sum_{j=1}^n ((g_1(x_{i,j}) + (1-g_1(x_{i,j}))q) \\
&\quad (1 - g_1(x_{i,j}) - (1-g_1(x_{i,j}))q)) \\
&= \frac{(\varepsilon N)^2}{n(1-q)^2} \left(\frac{n}{4} - \left(\frac{(1-q)y_i}{\varepsilon N} - \frac{(1-2q)\sqrt{n}}{2} \right)^2 \right) \\
&\le \frac{(\varepsilon N)^2}{4(1-q)^2}.
\end{aligned}$$

■

Thus, it is sufficient to set a constant q so that $\mathrm{Var}[Y_i] = O((\varepsilon N)^2)$. Since now each sampled (item, count) pair only consumes $O(\log(1/q)) = O(1)$ bits, the total cost is $O(\sqrt{n}/\varepsilon)$ bits.

Sampling with g_1, the general case.: The above simple scheme works when all $x_{i,j} \le \varepsilon N/\sqrt{n}$. When $x \ge \varepsilon N/\sqrt{n}$, $g_1(x)$ hits 1 and is no longer linear. So any $x_{i,j} \ge \varepsilon N/\sqrt{n}$ cannot be encoded in a Bloom filter, and unfortunately, there could be $O(\sqrt{n}/\varepsilon)$ of them, costing $O(\sqrt{n}/\varepsilon \cdot (\log u + \log N))$ bits. Smarter techniques are thus needed for the general case when the $x_{i,j}$'s take arbitrary values.

We write each $x_{i,j}$ in the form of

$$x_{i,j} = a_{i,j}\frac{\varepsilon N}{\sqrt{n}} + b_{i,j}, \tag{5}$$

where $a_{i,j}$ and $b_{i,j}$ are both non-negative integers and $a_{i,j} \le \frac{\sqrt{n}}{\varepsilon}$, $b_{i,j} < \frac{\varepsilon N}{\sqrt{n}}$. Note that

$$y_i = \frac{\varepsilon N}{\sqrt{n}} \sum_{j=1}^n a_{i,j} + \sum_{j=1}^n b_{i,j}. \tag{6}$$

The term $\sum_{j=1}^k b_{i,j}$ can be estimated using Lemma 5.1 since $b_{i,j} < \frac{\varepsilon N}{\sqrt{n}}$, so we focus on estimating the first term with variance $O((\varepsilon n)^2)$.

Our idea is to consider each $a_{i,j}$ in its binary form and dealing with each bit individually. Let $a_{i,j}[r]$ be the r-th rightmost bit of $a_{i,j}$ (counting from 0). For each r, node j encodes all the items i where $a_{i,j}[r] = 1$ in a Bloom filter with false positive probability q_r. Intuitively, q_r should be smaller for more significant bits (i.e., larger r), but we will derive this relationship later. For any item i, suppose $Z_{i,r}$ is the number of Bloom filters that asserts $a_{i,j}[r] = 1$. Below we show that

$$A_i = \frac{\varepsilon N}{\sqrt{n}} \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} 2^r \frac{Z_{i,r} - nq_r}{1 - q_r}$$

is an unbiased estimator for the first term of (6) and bound its variance.

Lemma 5.2: $\mathbf{E}[A_i] = \frac{\varepsilon N}{\sqrt{n}} \sum_{j=1}^{n} a_{i,j}; \mathrm{Var}[A_i] \le (\varepsilon N)^2 \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} 2^{2r} \frac{q_r}{1-q_r}$.

Proof: Let $c_{i,r} = \sum_{j=1}^{n} a_{i,j}[r]$. Since there are $n - c_{i,r}$ Bloom filters which may, with probability q_r, assert $a_{i,j}[r] = 1$ despite $a_{i,j}[r] = 0$, it is easy to see that $\mathbf{E}[Z_{i,r}] = c_{i,r} + (n - c_{i,r})q_r$, and $\mathrm{Var}[Z_{i,r}] = (n - c_{i,r})q_r(1-q_r) \le nq_r(1-q_r)$. Thus we have

$$\begin{aligned}\mathbf{E}[A_i] &= \frac{\varepsilon N}{\sqrt{n}} \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} 2^r \frac{\mathbf{E}[Z_{i,r}] - nq_r}{1-q_r} \\ &= \frac{\varepsilon N}{\sqrt{n}} \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} 2^r c_{i,r} = \frac{\varepsilon N}{\sqrt{n}} \sum_{j=1}^{n} a_{i,j},\end{aligned}$$

and

$$\begin{aligned}\mathrm{Var}[A_i] &= \frac{(\varepsilon N)^2}{n} \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} \frac{2^{2r}}{(1-q_r)^2} \mathrm{Var}[Z_{i,r}] \\ &\le (\varepsilon N)^2 \sum_{r=0}^{\log(\sqrt{n}/\varepsilon)} 2^{2r} \frac{q_r}{1-q_r}.\end{aligned}$$

■

So as long as we set $q_r \le 1/2^{3r+1}$, we can bound $\mathrm{Var}[A_i]$ by $O((\varepsilon n)^2)$, as desired. The cost for each $a_{i,j}[r] = 1$ is thus $O(\log(1/q_r)) = O(r)$ bits. Since each $a_{i,j}[r] = 1$ represents $2^r \frac{\varepsilon N}{\sqrt{n}}$ copies of an item, the amortized cost for every $\frac{\varepsilon N}{\sqrt{n}}$ copies is $O(r/2^r) = O(1)$ bits. Therefore, the total communication cost is $O(\sqrt{n}/\varepsilon)$ bits.

Theorem 5.3: When sampling with g_1, the communication cost can be made to $O(\sqrt{n}/\varepsilon)$ bits.

Sampling with g_2.: Because of its non-linear feature, the term $Y_{i,j}/g_2(Y_{i,j})$ will depend on the actual value of $x_{i,j}$ when sampling with g_2, so it is more difficult to save bits. In the following, we show how to combine g_1 and g_2 to reduce the communication cost to $O\left(opt(I) \log^2 \left(\frac{\sqrt{k}}{\varepsilon \cdot opt(I)}\right)\right)$ bits, which is better than using either g_1 alone or g_2 alone.

We observe that the estimator $Y_i = \sum_{j=1}^{n} \frac{Y_{i,j}}{g_2(Y_{i,j})}$ is itself another a frequency estimation problem, when we consider $\frac{Y_{i,j}}{g_2(Y_{i,j})}$ as the local count of item i at node j. So if Y_i estimates y_i well and we can estimate Y_i well, we will be estimating y_i well. To estimate Y_i, we simply run the sampling algorithm with g_1 on the $\frac{Y_{i,j}}{g_2(Y_{i,j})}$'s, i.e., we sample each $\frac{Y_{i,j}}{g_2(Y_{i,j})}$ with probability $g_1\left(\frac{Y_{i,j}}{g_2(Y_{i,j})}\right)$ and then encode the sampled items into Bloom filters as above.

Let T_i be the estimator thus obtained. It is clear that it is an unbiased estimator. Its variance, by the law of total variance, is

$$\begin{aligned}\mathrm{Var}[T_i] &= \mathbf{E}[\mathrm{Var}[T_i \mid Y_i]] + \mathrm{Var}[\mathbf{E}[T_i \mid Y_i]] \\ &\le O((\varepsilon N)^2) + (\varepsilon N)^2 = O((\varepsilon N)^2).\end{aligned}$$

Now we analyze the cost of this algorithm. First of all, since

$$\mathbf{E}\left[\sum_{i,j} \frac{Y_{i,j}}{g_2(Y_{i,j})}\right] = \mathbf{E}\left[\sum_i y_i\right] = N,$$

the $\frac{Y_{i,j}}{g_2(Y_{i,j})}$'s form exactly another instance of the frequency estimation problem with the same N, n, ε (albeit in expectation), so its cost is no more than $O(\sqrt{n}/\varepsilon)$ bits. To see the real improvement, the key observation is that overall there are only $opt(I)$ non-zero $\frac{Y_{i,j}}{g_2(Y_{i,j})}$'s, whereas there are much more non-zero local counts in the original problem. When encoding a non-zero $\frac{Y_{i,j}}{g_2(Y_{i,j})}$, we write it in the form of (5) and spend $O(r)$ bits for every $a_{i,j}[r] = 1$. This is $O(\log^2 a_{i,j})$ bits. (Note that we cannot use the charging argument as in Theorem 5.3 because here we need to bound the cost in terms of $opt(I)$.) Thus the total number of bits required is

$$\begin{aligned}\sum_{i,j,a_{i,j}\neq 0} \log^2 a_{i,j} &\le opt(I) \log^2 \left(\frac{\sum_{i,j} a_{i,j}}{opt(I)}\right) \\ &= opt(I) \log^2 \left(\frac{\sqrt{n}}{\varepsilon \cdot opt(I)}\right),\end{aligned}$$

where equality holds when all the $a_{i,j}$'s are equal.

Theorem 5.4: When sampling with g_2, the communication cost can be made to $O\left(opt(I) \log^2 \left(\frac{\sqrt{n}}{\varepsilon \cdot opt(I)}\right)\right)$ bits.

Remarks: When $n > 1/\varepsilon^2$, as argued in Section III we should sample the $\frac{Y_{i,j}}{g_2(Y_{i,j})}$'s by uniform sampling, which is better than using g_1. So the bound in the theorem above should be $O\left(opt(I) \log^2 \left(\frac{\min\{\sqrt{n}/\varepsilon, 1/\varepsilon^2\}}{opt(I)}\right)\right)$, to be more precise.

VI. SIMULATION RESULTS

We generated a data set with $u = 10,000$ items with their frequencies following the Zipf distribution, i.e., the i-th item has a frequency $y_i \propto 1/i$. We make the total count of all items to be $N = 10^9$. This represents a heavy-tail distribution that is typical in many real-world applications. Then for each item we randomly split its global count to $n = 1000$ nodes.

a) Sampling functions.: We We first compared the performance of the three sampling functions: $g_0(x) = x/(x+d)$ proposed in [21], and the two functions g_1 and g_2 proposed in this paper. The purpose of this comparison is two-fold: 1) we would like to see how large the gap is between worst-case optimality and instance optimality on a typical data set; and 2) directly sampling with these functions without using the techniques of Section V is extremely simple and this might be desirable in some implementations.

We have tested with varying error parameters (d for g_0 and ε for g_1, g_2) and plotted the results in Figure 1 (the three dashed lines). Each data point in the plot is the result of 100 repeated runs with the same parameter. The y-coordinate is the average cost (in terms of bytes, where each (item, count) pair is counted as 8 bytes) over the 100 runs. For each of the top-100 frequent items, we estimate its frequency and compute the variance of the 100 runs. Then we take the maximum variance of the 100 items as the x-coordinate. The reason for looking at the worst variance is that our goal is to estimate the frequency of *every* item reasonably well. From the plot we see that g_2 is generally 2 to 3 times cheaper than g_1 for achieving the same variance, and g_1 is 2 to 3 times cheaper than g_0 (note the log scale on y). So the difference is not as extreme as in the example we gave in the beginning, but we think we are happy to see this improvement on such a typical data set.

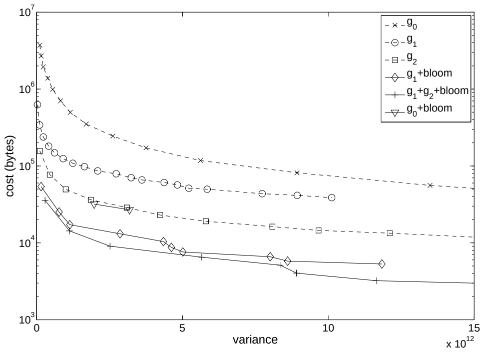

Fig. 1. Simulation results for different sampling functions (dashed lines) and for various algorithms based on Bloom filters (solid lines).

b) Combining with Bloom filters.: We have implemented our algorithm of using g_1 with Bloom filters and the algorithm of using g_1, g_2 together with Bloom filters. In the Bloom filters, we used the simple hash functions $h(x) = (ax^2 + bx + c) \mod p$ where p is a large prime, while each node uses random a, b, c generated independently.

The results are also plotted in Figure 1 (the solid lines). We can see that the use of Bloom filters has reduced the cost of g_1 by almost a factor of 10. As seen before, sampling with g_2 is more difficult to combine with Bloom filters, and as a result, the improvement is not as dramatic. The end result is that the two algorithms performed quite similarly. For instance, both of them transmitted $1.5 \times 10^4 \approx 15$K bytes to estimate the counts for the top-100 items with a maximum standard deviation of 10^6, which corresponds to $\varepsilon = 0.001$, a typical error as suggested in [6]. Note that the raw data has roughly $nu = 10^7$ (item, count) pairs, which amounts to 80M of data.

We also did our best-effort implementation of the algorithm of [21], which heuristically improves the bare use of g_0. We followed their description and optimized two sets of parameters. This gave the two data points in Figure 1, which are roughly 3 times worse than our algorithms. It is possible to tune the parameters to obtain other cost-variance tradeoffs, but it is doubtful that there exists a point on its tradeoff curve where it is better than ours.

We also generated data sets with different distributions to test the performance of the algorithms. We set the frequency of i-th item $y_i \propto (1/i)^\alpha$ (the general definition of the Zipf), then by varying the value of α, we get different distributions. The Simulation results for different α's are plotted in Figure 2. Here we also set $u = 10,000$ and $N = 10^9$, and for each item we randomly split its global count to $n = 1000$ nodes.

VII. Relative Errors

Here we briefly discuss how to guarantee a (p, ε)-error, i.e., estimating every $y_i \geq pN$ with variance $(\varepsilon y_i)^2$. First, it is known [11] that a uniform sample of size $O(1/\varepsilon^2 p)$ achieves this guarantee. Next we see if the general sampling framework can bring any improvement. Recall that the sampling function of [21], $g(x) = x/(x+d)$, has a variance of $\mathrm{Var}[Y_i] = dy_i$. We need $dy_i = (\varepsilon y_i)^2$, namely, $d = \varepsilon^2 y_i$, for all $y_i \geq pN$. So we need to set $d = \varepsilon^2 pN$, which implies that the cost could be $O(N/d) = O(1/\varepsilon^2 p)$.

Now consider the linear sampling function $g(x) = x/a$ for some a. From (3) in the proof of Theorem 3.1 we have $\mathrm{Var}[Y_i] = ay_i - y_i^2/n$. Thus we need $ay_i - y_i^2/n \leq (\varepsilon y_i)^2$, namely $a \leq (\varepsilon^2 + 1/n)y_i$ for all $y_i \geq pN$. So it suffices to set $a = (\varepsilon^2 + 1/n)pN$, which means that the cost is $O(N/a) = O\left(\frac{1}{(\varepsilon^2+1/n)p}\right)$. This, again, is better than $O(1/\varepsilon^2 p)$ when $n \ll 1/\varepsilon^2$.

However, the sampling function g_2 is tailored for the absolute error. It remains an interesting problem to see if there is an instance-optimal sampling function for (p, ε)-errors.

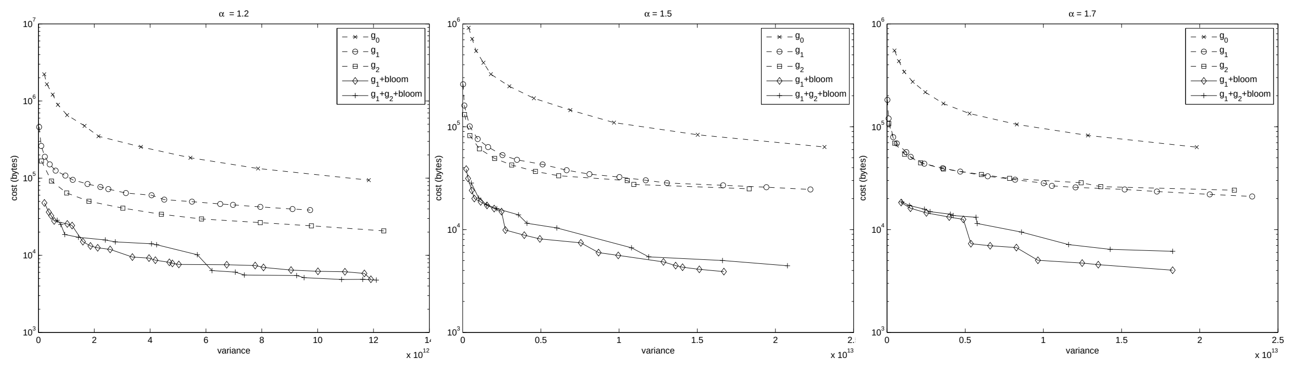

Fig. 2. Simulation results for different α's.

VIII. Remarks and Open Problems

In this paper we have designed worst-case optimal and instance-optimal sampling algorithms for the distributed frequency estimation problem. However, we need to emphasize that the optimality of our algorithms holds only within the sampling framework defined in Section I. Theoretically, it is an intriguing open question to determine the (worst-case) complexity of the problem with no restrictions on the way how the algorithm works. This could be a difficult problem, since even the deterministic complexity is not understood yet. Note that, however, instance-optimality will not be possible to achieve if we do not have any restrictions on the algorithm, since we can design a "wild guessing" algorithm that just outputs the result directly if the input is the one being guessed, while falls back to a naive algorithm on all other inputs. No reasonable algorithm can beat this algorithm on that particular input.

References

[1] http://en.wikipedia.org/wiki/bloom_filter.

[2] B. Aronov, S. Har-Peled, and M. Sharir. On approximate halfspace range counting and relative epsilon-approximations. In *Proc. Annual Symposium on Computational Geometry*, pages 327–336, 2007.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2003.

[4] P. Cao and Z. Wang. Efficient top-k query calculations in distributed networks. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2004.

[5] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2005.

[6] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. In *Proc. International Conference on Very Large Data Bases*, 2008.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), 2008.

[9] R. Fagin, A. Lotem, and M. Noar. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.

[10] M. Greenwald and S. Khanna. Power conserving computation of order-statistics over sensor networks. In *Proc. ACM Symposium on Principles of Database Systems*, 2004.

[11] Y. Li, P. M. Long, and A. Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 62:516–527, 2001.

[12] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. International Conference on Very Large Data Bases*, 2002.

[13] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 2006.

[14] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. In *Proc. International Conference on Very Large Data Bases*, 2005.

[15] B. Patt-Shamir and A. Shafrir. Approximate distributed top-k queries. *Distributed Computing*, 21:1–22, 2008.

[16] E. Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. *Computer Science–Theory and Applications*, 5675:263–273, 2009.

[17] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. International Conference on Embedded Networked Sensor Systems*, 2004.

[18] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[19] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *Proc. ACM Symposium on Principles of Database Systems*, 2009.

[20] H. Zhao, A. Lall, M. Ogihara, and J. Xu. Global iceberg detection over distributed data streams. In *Proc. IEEE International Conference on Data Engineering*, 2010.

[21] Q. Zhao, M. Ogihara, H. Wang, and J. Xu. Finding global icebergs over distributed data sets. In *Proc. ACM Symposium on Principles of Database Systems*, 2006.