Estimating Simple Functions on the Union of Data Streams

Phillip B. Gibbons* Srikanta Tirthapura†

November 10, 2000

Abstract

Massive data sets often arise as physically distributed data streams. We present algorithms for estimating simple functions on the union of distributed data streams, while using only logarithmic space per stream. Our algorithms employ a novel *coordinated sampling* technique to extract a sample of the union; this sample can be used to estimate aggregate functions on the union. The technique can also be used to estimate aggregate functions over the distinct "labels" in one or more data streams, e.g., to determine the zeroth frequency moment (i.e., the number of distinct labels) in one or more data streams. Our space and time bounds are the best known for these problems, and our logarithmic space bounds for *coordinated* sampling contrast with polynomial lower bounds for *independent* sampling. We relate our distributed streams model to previously studied non-distributed (i.e., merged) streams models, presenting tight bounds on the gap between the distributed and merged models for deterministic algorithms.

1 Introduction

This paper considers the following distributed setting for massive data sets. There are two streams $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of data items. Alice observes the $\{a_i\}$ in order, and Bob observes the $\{b_i\}$ in order. Alice and Bob each have some limited amount of workspace, $w \ll n$ bits, to be used while observing their respective streams. After observing their streams, Alice and Bob send the contents of their workspace to a Referee, who estimates a function F on A and B, without seeing either stream. Note that Alice and Bob are not allowed to communicate with each other directly. An example function is given in Figure 1. We are interested in minimizing

(1) the total workspace used by Alice and Bob, and
(2) the time taken by Alice and Bob to process each data item.

More generally, we have $t \geq 2$ parties, each observing their respective data streams, and a Referee estimating a function on the streams. We denote this setting as the *distributed streams* model.

This model is motivated by the monitoring and characterization of internet traffic. Monitoring devices in the network observe a stream of packets. Each device has a small workspace in which to store information on its observed stream, and the contents are periodically sent to a central data analyzer, in order to compute aggregated statistics on the streams. This set-up is used, e.g., in Lucent's InterpretNet and Cisco's NetFlow network monitoring products.

The distributed streams model combines features of both streaming models (e.g., [AMS96, GM99, HRR98, FKSV99a, FKSV99b, FS00, Ind00, GMMO00]) and communication complexity models (e.g., [KN97, KNR99]). As in streaming models (but not communication complexity models), the data is observed once, and we seek to minimize the workspace and the time to process each data item. However, more in line with *simultaneous,*

*Information Sciences Research Center, Bell Laboratories, Lucent Technologies, Murray Hill NJ 07974-0636. Email: gibbons@research.bell-labs.com

†Computer Science Department, Brown University, Providence, RI 02912-1910. Email: snt@cs.brown.edu

Input to Alice: $A = \{a_1, \ldots, a_n\}$, a stream of bits.
Input to Bob: $B = \{b_1, \ldots, b_n\}$, a stream of bits.
Referee to estimate: $U(A, B) = \sum_{i=1}^{n} (a_i \vee b_i)$.

Application: The bits correspond to the characteristic vectors of two sets A^* and B^* over the same domain of size n. The function $U(A, B) = |A^* \cup B^*|$.

Figure 1: The union function U

1-round communication complexity models [KNR99, NS96] and sketch models (e.g., [BCFM98, AGMS99, FKSV99a]), the input is shared among multiple players who communicate only by sending a message to a referee, who then computes/estimates the function.

Often, it is important to consider aggregate functions on the *union* of the data streams. For example, the same logical stream of data from a source to a destination in a virtual private network is frequently split among multiple paths in the network, in order to improve throughput and reliability. Network monitoring devices along each path observe only the divided traffic. Yet, to understand the logical stream, the characterization must be done on the *union* (not the sum) of the constituent streams. More generally, a natural partitioning of the data items exists, e.g., by (discretized) time, by source/destination, etc., such that items within the same partition must be specially aggregated across all streams.

In this paper, we study the problem of estimating simple functions on the union of streams. We show how a technique we call *coordinated 1-sampling* can be effectively used to estimate simple monotone functions in the distributed streams model.

We demonstrate the technique by focusing first on the union function U in Figure 1, in which each data item a_i and b_i is a single bit and $U(A, B) = \sum_{i=1}^{n} (a_i \vee b_i)$ is the number of 1's in the bitwise OR of the two streams. Exact computation of U would solve the set disjointness problem, and hence requires $\Omega(n)$ workspace, even for randomized algorithms [KN97]. Because for massive data streams, linear workspace is unacceptably large (see, e.g., [GM99]), we instead seek to approximate U to within a small relative error. An (ϵ, δ)-*approximation scheme* for a quantity X is a (randomized) procedure that, given any positive ϵ and δ, computes an estimate $\hat{X}$ of X that is within a relative error of ϵ with probability at least $1 - \delta$, i.e., $\mathbf{Pr}\left\{|\hat{X} - X| \leq \epsilon X\right\} \geq 1 - \delta$.

We present two (ϵ, δ)-approximation schemes for U that use $O(\frac{\log(1/\delta) \log n}{\epsilon^2})$ workspace. The first is for the *public coins* model [KNR99], in which Alice and Bob have access to the same random string of unbiased and fully independent bits. This scheme is the simpler of the two, and requires only constant time to process each data item. Note that in practice, the public coins in the algorithm can be simulated by having Alice and Bob use the same pseudo random number generator, with the same starting seed of $\Theta(\log n)$ bits. We will refer to this setting as the *distributed streams model with public coins*.

Our second approximation scheme is for a *stored coins* model, in which Alice and Bob have a shared random string of unbiased and fully independent bits, but these bits must be stored at Alice and Bob prior to observing their inputs. The space to store these bits must be accounted for in their workspace bound. This scheme is more involved, as we need to provide an explicit construction by each party of the needed random bits from a small random string, and analyze the scheme under the limited independence of the constructed bits. The time to process each data item is dominated by the time required to perform $O(\log(1/\delta))$ multiplications over a finite field of $\Theta(\log n)$ bits. We will refer to this setting as the *distributed streams model with stored coins*. Previous works on streaming models (e.g., [AMS96, HRR98, FKSV99a, FKSV99b, FS00, Ind00]) have studied settings with stored coins. Stored coins differ from private coins (e.g., as studied in communication complexity [KNR99, New91, NS96]) in that the same random string can be stored in all parties.

We show that our sampling technique could be of wider applicability in streaming models by using it in in an (ϵ, δ)-approximation scheme for F_0 (the number of distinct elements) of a sequence. Given a sequence

$A = \{a_1, \ldots, a_n\}$ where each item a_i is a member of $\{1, \ldots, m\}$, we show an (ϵ, δ)-approximation scheme for F_0 whose space complexity is $O(\frac{\log(1/\delta) \log m}{\epsilon^2})$ and the time required to process each item is dominated by the time required to perform $O(\log(1/\delta))$ multiplications over a finite field of $\Theta(\log m)$ bits. We note that this is an interesting result in itself because of the importance of the F_0 function in database optimization (see, e.g., [HNSS95]) and in IP traffic analysis, e.g., the number of distinct web pages requested, or the number of distinct visitors to a website. This problem has been studied in [FM85, AMS96], but we do not know of an (ϵ, δ)-approximation scheme for F_0 whose bounds match the bounds we obtain.

Our logarithmic space bounds for union and for F_0 using coordinated sampling contrast with an $\Omega(\sqrt{n})$ lower bound for union and an $\Omega(m)$ lower bound for F_0 using an independent sample drawn from each stream.

We show how to extend our scheme to the scenario studied in [FKSV99a, Ind00] and elsewhere in which data items are (label,value) pairs. Unlike previous work on data streams, our sampling technique obtains a random sample of the (distinct) labels in a stream (or in the union of streams) of a desired target size β in $O(\beta)$ workspace, and could be used to estimate other functions which are well-estimated with a sample of size β. Moreover, for labels in the sample that are in more than one stream, the referee learns the respective values from each stream. Thus we can estimate aggregate functions over the values associated with distinct labels in one or more data streams, e.g., the variance in request sizes averaged over all distinct destinations. As epitomized by the TPC benchmarks [TPC00], which are the primary industry benchmarks for large scale query processing, many queries and reports seek aggregates of values over the distinct "labels" of a data set.

Next, we consider the previously-studied streams model in which there is only one party, who observes both streams, and the streams are interleaved in an arbitrary order by an adversary (e.g., [FKSV99a, Ind00]). We consider the obvious generalization of this model to $t > 2$ streams, and refer to the model as the *merged streams* model. We present a comparison of the relative power of the distributed and merged streams models. We show that for any function f, the deterministic merged stream complexity (space bound) is within a factor of t of the deterministic t-party distributed stream complexity, and that this gap is existentially tight. It follows that deterministic merged streams algorithms can be designed assuming that the streams are not interleaved, at a penalty of at most t (which is viewed as a small constant). The comparison for randomized complexities remains open. In fact, we show that a related previous result on the randomized communication complexity for boolean functions [KNR99] suffers from a flaw in the proof, thereby reopening that related problem.

The remainder of the paper is organized as follows. Section 2 presents comparisons with previous work. Section 3 presents our coordinated 1-sampling technique and its application in computing the union function U. Extensions to the basic method and its applications are discussed in Section 4. Our results comparing distributed and merged stream models are in Section 5, with conclusions following in Section 6.

2 Preliminaries and Related Work

Previous work on data streams has studied

- approximating a function such as the kth frequency moment on a single stream observed by a single party (e.g., [AMS96, HRR98, AGMS99, GM98]);
- approximating a function such as the L^p-difference on two streams observed by a single party (e.g., [FKSV99a, FKSV99b, FS00, Ind00]); and
- computing sketches of t data sets, each observed by a distinct party, in order to approximate a function such as the set resemblance or the join size (e.g., [BCFM98, AGMS99, FKSV99a, Ind00, IKM00]).

The small space data structures studied in these works are examples of *synopsis data structures*; frameworks for studying synopsis data structures beyond the data stream context were presented in [GM99], along with a survey of results.

As has been observed in many of these works, algorithms designed for one of the above data stream scenarios are often suited for the other scenarios as well. The same can be said for the distributed streams model: For example, an (ϵ, δ)-approximation scheme for a function f in the distributed streams model trivially implies an (ϵ, δ)-approximation scheme for f in the sketch model, with the same space bound. On the other hand, algorithms designed for a single party are sometimes ill-suited for the distributed streams model, e.g., the kth frequency moment ($k \geq 3$) algorithm of [AMS96] requires counting at Bob the number of occurrences of labels known only to Alice. This paper presents results relating the complexity of the various streams models.

Union. In the distributed streams model, $\sum_i a_i$ and $\sum_i b_i$ can be trivially computed exactly and deterministically by Alice and Bob, respectively, using $\log n$ bits each. Thus $\frac{4}{3}$ times the maximum of these two is an approximation for the union, with space complexity $\log n$ and relative error $\epsilon = \frac{1}{3}$. For two streams, the identity $2|\sum_i (a_i \vee b_i)| = |\sum_i a_i| + |\sum_i b_i| + |\sum_i (a_i \triangle b_i)|$ can be used to obtain an (ϵ, δ)-approximation scheme for U for the distributed streams model with stored coins from the (ϵ, δ)-approximation scheme for the size of the symmetric difference, $|\sum_i (a_i \triangle b_i)|$, in [FKSV99a]. The symmetric difference algorithm is presented for the merged streams model, but extends to distributed streams. The resulting scheme uses the same space bound as ours, but the time to process each data item is worse: it is dominated by the time required to perform $O(\log(1/\delta)/\epsilon^2)$ multiplications over a finite field of $\Theta(\log n)$ bits.[1] Our algorithm uses completely different techniques and is considerably simpler. Moreover, while our scheme extends trivially to more than two streams, there is no approach known for extending their scheme to estimate the union function U of $t > 2$ streams.

A crucial factor in our approach is that Alice and Bob's sampling is coordinated:

Lemma 1 *Estimating U within $\epsilon < \frac{1}{3}$ with probability $> \frac{1}{2}$ by sampling independently from Alice and from Bob requires $\Omega(\sqrt{n})$ sample sizes.*

Thus uncoordinated sampling would require $\Omega(\sqrt{n})$ workspace.

Distinct Counting. Consider the zeroth frequency moment (F_0) of a sequence of n items in $\{1, \ldots, m\}$, where $m \leq n$. This function has been studied in the context of a single stream for both public coins [FM85] and stored coins [AMS96]. These previous algorithms trivially extend to the distributed streams model, but we do not know how to convert them into an (ϵ, δ)-approximation scheme; and very likely, even if we obtained one, the space and/or time bounds would be worse than our algorithm's bounds.[2] The previous algorithms use probabilistic counting instead of sampling: only the estimate is obtained, not a sample that can be used for multiple estimation problems. Note that computing the union U is simply a special case of computing F_0, in which all items within a stream are distinct. Coordinating the sampling is even more crucial for F_0:

Lemma 2 (follows from [CMN98, CCMN00]) *Estimating F_0 within a small constant ϵ with probability $> \frac{1}{2}$ by taking independent samples of both streams requires $\Omega(m)$ space.*

Communication complexity. In communication complexity models [KN97], the parties have unlimited time and space with which to process their respective inputs. One-way communication complexity results can often be related to the merged streams model, whereas simultaneous, 1-round communication complexity results can often be related to the distributed streams model. Both public/common and private coin communication complexity models have been studied (e.g., [New91, NS96]). For streaming models and their applications,

[1] Note that the time per item is critical in practice, due to the extremely high network traffic rate.

[2] [AMS96] computes Y such that for any $c > 2$, $\mathbf{Pr}\left\{\frac{F_0}{c} \leq Y \leq cF_0\right\} > 1 - \frac{2}{c}$. Thus the estimate obtained is not even guaranteed to be within a factor of 2, and the success probability is only constant. The paper points out that "it is possible to improve the accuracy and the success probability of the algorithm by increasing the space it uses", but provides no details. The success probability can be readily made to be δ, by taking the median of $\log(1/\delta)$ such estimates. On the other hand, it is not clear how to obtain an ϵ relative error from this approach (e.g., by averaging many estimates), in small space and time, due to the max function and exponentiation in the approach. In particular, approaches based on computing many such estimates would multiply the time by the number of estimates. The time bound would be at least a factor of $1/\epsilon^2$ worse than our approach.

the dichotomy is between public coins and stored (common) coins; this dichotomy does not arise in communication complexity, due to the absence of workspace constraints.

3 Coordinated 1-Sampling

Our algorithm for estimating U is based on the following random sampling procedure. Given a p between 0 and 1, Alice and Bob decide on a randomly chosen subset $S(p)$ of $\{1, \ldots, n\}$ as the positions at which to sample their bit streams. $S(p)$ is created by selecting each number in $\{1, \ldots, n\}$ independently with probability p. Both Alice and Bob know $S(p)$. Alice and Bob sample their streams at positions specified by $S(p)$ and store only those positions where the 1's occurred. Alice collects $\{i | i \in S(p) \wedge a_i = 1\}$ and Bob $\{i | i \in S(p) \wedge b_i = 1\}$. Our estimate for U is $Y = \frac{\sum_{i \in S(p)} (a_i \vee b_i)}{p}$. By Chernoff bounds, $\mathbf{Pr}\{Y \notin (1-\epsilon, 1+\epsilon)U\} < 2\exp(-Up\epsilon^2/2)$. By storing only the 1's, the expected space used is $O(U \cdot p \log n)$, because the positions of an expected Up items have to be stored (instead of np items), and storing each of them takes $\log n$ bits.

If we choose the sampling probability p such that the product Up is about $\frac{\log(1/\delta)}{\epsilon^2}$, then we are assured of getting a "good" estimate with a high probability and the space complexity is also low. The problem, of course, is that Alice and Bob do not know U and hence cannot decide on the right value of p beforehand. To solve this, we adapt the sampling probability as the algorithm proceeds. Each party starts off by sampling with probability 1. If the current sampling probability at a party causes the stored sample to become too large, then the probability is halved and the sampling proceeds. This way, we ensure that the space taken by the sample is never too large, and also, as we prove later, the accuracy of our estimate meets the required criteria. Two questions arise here:

1. How does a party switch to a lower probability of sampling? When we decide to do that, some bits of the stream have already flown by, and we cannot observe them anymore. We must ensure that these bits are not needed in our sample. To ensure this, we select $S(p/2)$ from $S(p)$ by including each element with probability $\frac{1}{2}$. Thus all the sample points required by the new sample, $S(p/2)$, up through the current data stream item are already there in the stored sample associated with $S(p)$.

2. What if the two parties are sampling their respective streams at different probabilities and hence, at different positions? This is entirely possible since each party makes the decision to change the sampling probability independently of the other, based on the number of 1's in their stream in the selected positions. But we know the following: If Alice ends at probability p_A and Bob at p_B, and say $p_A \geq p_B$, then $S(p_A) \supset S(p_B)$. So, the referee can determine the sample of the sequence $\{a_i \vee b_i\}$ resulting from a sampling probability of p_B. A key property we are exploiting here is that U is monotone, so that when Alice targets $p \cdot \sum a_i \approx \frac{\log(1/\delta)}{\epsilon^2}$ and Bob targets $p \cdot \sum b_i \approx \frac{\log(1/\delta)}{\epsilon^2}$, we have that $p \cdot U \approx r\frac{\log(1/\delta)}{\epsilon^2}$ for $1 \leq r \leq 2$.

3.1 Public Coins Scheme

We now present our algorithm for public coins. We assume that Alice and Bob have access to a hash function $e : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, d = \log n\}$ such that for each $i \in \{1, 2, \ldots, n\}$, $e(i)$ is a random variable distributed as $\mathbf{Pr}\{e(i) = j\} = 1/2^j, j = 1 \ldots d-1$. We further assume that these random variables are i.i.d.[3] In the next section, we show how we relax these assumptions, at a cost to the processing time per bit. The random sample at probability $1/2^l$ is the set of indices which have been hashed to a value greater than or equal to l, i.e., $S(1/2^l) = \{i | e(i) \geq l\}$.

We describe the algorithm for Alice. (The algorithm for Bob is the same.) At any stage in the execution, Alice is at a particular "level" l which is related to her current sampling probability. At level l, she is sampling

[3]Technically, we need to access the ith $\log n$ bit word w in the public random string of uniform bits, and then compute $e(i)$ as the largest j such that the j most significant bits of w are all 0.

Alice: *Initialization:* $l \leftarrow 0$, $S \leftarrow \emptyset$.
Upon receiving a_i*:*

- If $(a_i = 1)$ and $(e(i) \geq l)$, then $S \leftarrow S \cup \{(i, e(i))\}$ // Store the positions of 1's in S. Position i is in the sample until the value of l exceeds $e(i)$.
- If $|S| > c\alpha$ then // Sample size is too large, shift to a higher level, i.e., a lower sampling probability.
  - $l \leftarrow l + 1$
  - $S \leftarrow \{(i, l_i) | l_i \geq l\}$ // Discard sample points that do not belong to the current level.

Finally: Send l and S to the referee.

Referee: The referee receives l_A, S_A from Alice and l_B, S_B from Bob. From these, the referee knows a_i for all i in $S(1/2^{l_A})$ and b_i for all i in $S(1/2^{l_B})$. Let $l^* = \max(l_A, l_B)$. The referee outputs $2^{l^*} \cdot \sum_{i \in S(1/2^{l^*})}(a_i \vee b_i)$.

Figure 2: Public coins algorithm.

with a probability of $\frac{1}{2^l}$. She maintains the set $\{i | (a_i = 1) \wedge (i \in S(1/2^l))\}$, thus implicitly maintaining the value of a_i for every position in $S(1/2^l)$. She starts at level 0; the level may increase as the computation proceeds. S is her sample. Let $\alpha = \frac{\log(3/\delta)}{\epsilon^2}$, and let $c = 84$, a constant determined from the analysis. The algorithm is given in Figure 2.

Theorem 1 *For any positive $\epsilon \leq 1$ and $\delta \leq 1$, the above algorithm is an (ϵ, δ)-approximation scheme for U, the size of the union of two bit streams, in the distributed streams model with public coins. The algorithm uses $O(1)$ worst case expected time to process each item, and a total of $O(\frac{\log(1/\delta) \log n}{\epsilon^2})$ bits of workspace.*

Proof. *Space Complexity:* Alice and Bob each store up to $c\alpha$ pairs of $(i, e(i))$. Hence, the space complexity is $O(\alpha \log n)$, which is $O(\frac{\log(1/\delta) \log n}{\epsilon^2})$.

Time Complexity: Upon receiving a bit a_i, Alice stores i if $e(i)$ is greater than the current level, l. If storing the bit requires Alice to move to a "higher" level, then she incurs the cost of removing all the items in the sample that do not belong to the next level. The expected number of level changes that any item $a_i = 1$ survives is two. Amortized over n bits, this gives an amortized expected cost of three operations, one of them being the computation of e. By applying "lazy discarding" when changing levels, so that once $|S|$ reaches $c\alpha$, it remains there, the amortized bound becomes a worst case bound.

We omit the correctness proof for this model due to page constraints. The proof is similar in spirit, but simpler, than the proof for the stored coins model given below. ∎

3.2 Stored Coins Scheme

The algorithm for the stored coins model is identical to the algorithm for public coins except for the following differences: We can no longer afford (in terms of workspace) an all powerful hash function e that generates fully independent random variables. Instead, we settle for one that generates random variables that are pairwise independent. But each instance gives us an estimate that is within an ϵ factor of the actual value with probability greater than $\frac{2}{3}$. In order to get the error probability down to δ, we take the median of the values computed by $s_2 = 48 \log(\frac{1}{\delta})$ instances of the algorithm. Interestingly, the threshold sample size at which the algorithm shifts to a higher level is smaller here than in the public coin algorithm: it is c/ϵ^2, for $c = 72$, a constant determined by the analysis. Let $\alpha = 1/\epsilon^2$.

Let e_k denote the hash function used by instance k of the algorithm. For every k, e_k is a mapping from $\{1 \ldots n\}$ to $\{0 \ldots d = \log n\}$, such that for $0 \leq l \leq (d-1)$, $\mathbf{Pr}\{e_k(i) = l\} = \frac{1}{2^{l+1}}$. $e_k(i)$ is computed as

follows: we consider the numbers $\{1 \ldots n\}$ as members of the field $G = GF(2^d)$. In a preprocessing step, we choose q_k and r_k uniformly and independently at random from G and store them with Alice and Bob. In order to compute an $e_k(i)$, Alice (Bob) computes $x = q_k \cdot i + r_k$, all operations being performed in G. We represent x as a d-bit vector and then $e_k(i)$ is the largest j such that the j most significant bits of x are zero (i.e., $j = d - \lfloor \log x \rfloor - 1$). Clearly $e_k(i)$ is a number between 0 and d. The two properties of e_k that we use are: (1) x is distributed uniformly over G. Hence the probability that $e_k(i)$ equals l (where $l < d$) is exactly $\frac{1}{2^{l+1}}$. (2) The mapping is pairwise independent, i.e., for distinct i and j, $\mathbf{Pr}\{(e_k(i) = k_1) \wedge (e_k(j) = k_2)\} = \mathbf{Pr}\{e_k(i) = k_1\} \cdot \mathbf{Pr}\{e_k(j) = k_2\}$.

Theorem 2 *For any positive $\epsilon \leq 1$ and $\delta \leq 1$, the above algorithm is an (ϵ, δ)-approximation scheme for U, the size of the union of the two bit streams, in the distributed streams model with stored coins. The algorithm uses a worst case expected $O(\log(1/\delta))$ finite field operations to process each item, and a total of $O(\frac{\log(1/\delta) \log n}{\epsilon^2})$ bits of workspace.*

Proof. Each of the s_2 instances of the algorithm stores up to $c\alpha$ pairs, and also requires storing q_k and r_k, for a total of $O(\frac{\log(1/\delta) \log n}{\epsilon^2})$ space. We store the sample S_k as an array $S_k[1 \ldots d]$ of linked lists, such that list $S_k[j]$ contains all items in the sample that die at level j. The proof of the time complexity bound is left for the appendix. The proof of correctness is given next. ∎

Proof of correctness. A difficulty in analyzing our algorithm is that Alice and Bob decide when to stop changing levels based on the outcome of random trials, and hence may stop at incorrect levels, and make correspondingly bad estimates.

Let 1_A denote the set $\{i | a_i = 1\}$, i.e., the positions of 1's in Alice's stream. Similarly, let $1_B = \{i | b_i = 1\}$, and let $1_U = \{i | a_i = 1 \vee b_i = 1\}$. The referee is trying to estimate the size of 1_U. The random variable computed by the kth instance of the algorithm, z_k, is described with the help of the following process: We place the numbers $1 \ldots n$ in "levels" $\{0 \ldots d\}$ by placing i in every level from 0 through (and including) $e_k(i)$. For $l \in \{0, \ldots d\}$ and $i \in \{1, \ldots n\}$, we define the random variables X_{li}. $X_{li} = 1$ if i was placed in level l and 0 otherwise. For every level $l \in \{0 \ldots d\}$, we define $X_l = \sum_{i \in 1_U} X_{li}$ and $X_l^A = \sum_{i \in 1_A} X_{li}$ and $X_l^B = \sum_{i \in 1_B} X_{li}$. Note that X_l is greater than or equal to both X_l^A and X_l^B.

Here's how the algorithm fits into this process: Alice stops at level l_A where l_A is the lowest numbered level l such that $X_l^A < c\alpha$. Similarly, Bob stops at level l_B. If l_A or l_B equal d, the algorithm quits. Otherwise it returns the estimate $z_k = 2^f \cdot X_f$ where $f = \max\{l_A, l_B\}$. For every level $l \in \{0 \ldots d-1\}$, we define: $B_l = 1$ if $2^l X_l \notin [(1-\epsilon)U, (1+\epsilon)U]$ and 0 otherwise. Level l is "bad" if $B_l = 1$, and "good" otherwise. Let S_i denote the event that the algorithm stops in level i, i.e., that f equals i.

Theorem 3 *The probability that the kth instance of the algorithm fails to produce an estimate which is within a factor ϵ of U is less than $\frac{1}{3}$.*

Proof. This fails to do so in the following cases:

- l_A or l_B equals d, i.e., $f = \max\{l_A, l_B\} = d$ and the algorithm quits.
- f is less than d, but f is a bad level, i.e., for some level k, the events S_k and B_k are both true.

Let P denote the probability that the algorithm fails. P is less than the sum of the probabilities of the above events. $P \leq \mathbf{Pr}\left\{2^f X_f \notin ((1-\epsilon)U, (1+\epsilon)U)\right\} + \mathbf{Pr}\{S_d\} = \sum_{i=0}^{d-1} \mathbf{Pr}\{S_i \wedge B_i\} + \mathbf{Pr}\{S_d\}$

Let level l denote the first level such that $E[X_l] < C\alpha$, for $C = 36$, a constant determined from the analysis. Note that l is less than d, since $E[X_d] = \frac{U}{2^d} = \frac{U}{n} \leq 1 < C\alpha$. We split the above sum as follows: $P = \sum_{i=0}^{l} \mathbf{Pr}\{S_i \wedge B_i\} + \sum_{i=l+1}^{d-1} \mathbf{Pr}\{S_i \wedge B_i\} + \mathbf{Pr}\{S_d\} \leq \sum_{i=0}^{l} \mathbf{Pr}\{B_i\} + \sum_{i=l+1}^{d-1} \mathbf{Pr}\{S_i\} + \mathbf{Pr}\{S_d\}$ $= \sum_{i=0}^{l} \mathbf{Pr}\{B_i\} + \sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\}$.

The idea here is that the first few levels (until l) are likely to have good estimates and it is unlikely that we stop in a later level ($l+1$ onwards). We state the following two lemmas, which together prove Theorem 3. The proofs of these lemmas are provided in the appendix. ∎

Lemma 3 $\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < 1/9$

Lemma 4 $\sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\} < 1/9$

From Theorem 3, and Chernoff bounds, we have:

Lemma 5 *The median of the set* $\{z_k | k = 1, \ldots, s_2\}$ *is within an* ϵ *relative error of* U *with probability* $\geq 1 - \delta$.

4 Extensions and Applications

In this section, we sketch only a few of the extensions and applications of coordinated 1-sampling.

The algorithms of the previous section can be extended in the following three ways, without increasing the per stream space bound or the per item time bound:

1. The same algorithm can be used for $t > 2$ parties.
2. The same algorithm can be used when the bits do not arrive in the same order to Alice and Bob. In this scenario, Alice observes pairs (i, a_i) and Bob observes pairs (j, b_j), where an adversary controls the order of the pairs in each data stream.
3. The same algorithm can be used when only the 1-bits appear in the data stream: Alice and Bob each observe an unordered sequence of the positions of the 1-bits in their respective streams. The time bounds in Theorems 1 and 2 are per 1-bit. The space bounds have a factor that is a logarithm of the domain size for position numbers.

Unlike previous work on data streams, our coordinated 1-sampling approach obtains a random sample of the union, of a target size β in $O(\beta)$ workspace, along with a scaling factor.

Estimating F_0 and related functions. The approach can be extended to obtain a sample over distinct labels, even when a label can appear multiple times within a stream. In this scenario, we would store the $c\alpha$ labels present in our sample in a hash table of size $\Theta(\alpha)$. This enables constant expected look-up time. Associated with each label in the sample will be an accumulation value relevant to the function being estimated, e.g., the number of occurrences of the label within the stream. For estimating F_0, there is no accumulation value, and the hash table is used merely to ensure that each distinct label is stored only once in the sample at a party. Maintaining a distinct labels sample in the presence of new data is useful for approximate query answering systems for data warehouses, such as the Aqua system [AGPR99a, AGPR99b]. As discussed in Section 1, industry benchmarks have many queries and reports over distinct values.

Average interarrival gap. Our relative error approximation of U permits a relative error approximation of the average interarrival gap G in the union of multiple streams, for the common case where time is discretized, i.e., $G = \frac{\text{number of time slots}}{\text{size of union}}$.

Resemblance. Coordinated 1-sampling can be used to approximate the set resemblance. We avoid the complications of the min-wise permutations used in [BCFM98], by using powers of 2 sampling.

General values. Our algorithm can also be generalized for the case when the a_i's and b_i's are not binary, but are integers in the range $\{0, \ldots, M-1\}$, and the function to be estimated is $g(A, B) = \sum_{i=1}^{n} \max\{a_i, b_i\}$. In the full paper, we show how to obtain the same asymptotic space bound as the boolean case (as long as M is at most polynomial in n), at the cost of the time bound. Let $Y(x, M)$ (where $0 \leq x < M$) be the unary

representation of x, with enough zeros added to the right to make the length of the representation exactly M. For example, $Y(5,8) = 11111000$. We observe that $U(Y(x,M), Y(y,M)) = \max(x,y)$, where $x, y < M$. Hence, we treat the integer stream $A = \{a_i | i = 1, \ldots, n\}$ as a bit stream $Y(A)$ formed by concatenating the bit streams $\{Y(a_i, M) | i = 1, \ldots, n\}$ and similarly for B, and then apply the binary input algorithm.

5 Comparison of models

In this section we compare the power of different streaming models on distributed data sets, and use ideas from communication complexity to derive relationships between these models for deterministic algorithms.

The *deterministic merged stream complexity* of a function f, denoted by $DM_t(f)$, is the minimum workspace required by a deterministic algorithm to compute the function $f(X_1, \ldots, X_t)$ when t streams $X_1, \ldots, X_t$ are observed by a single party, but interleaved arbitrarily. Similarly, the *deterministic distributed stream complexity* of f, denoted by $DD_t(f)$ is the minimum workspace required by a deterministic algorithm to compute the function $f(X_1, \ldots, X_t)$ in the distributed streams model. In both the distributed and merged stream models, arbitrary pre- and post-processing resources (both space and time) are permitted, and only the resources used while observing the streams are counted towards the complexity.

Theorem 4 *For any $t \geq 1$ and any function f, (1) $DM_t(f) \leq DD_t(f)$; (2) $DD_t(f) \leq t \cdot DM_t(f)$; and (3) the factor of t bound is existentially tight, i.e., there exists a function g for which $DD_t(g) = t \cdot DM_t(g)$.*

Proof. (1) In the merged streams model, we can simulate the independent parties of a distributed streams protocol, within the same space bounds.

(2) Given any protocol P_m in the merged stream model which can compute f in space s, we will show a protocol P_d in the distributed stream model which can compute f in space $t \cdot s$. Suppose all of stream X_1 was given to P_m before any item of another stream. Let $h_1(X_1)$ denote the "sketch" of X_1 computed by P_m after it has seen X_1. Similarly, we have $h_2(X_2), \ldots, h_t(X_t)$. In P_d, the ith party computes $h_i(X_i)$. The space complexity of this algorithm is $t \cdot s$. Consider the "truth table" of f, which is a table of size $\prod_{i=1}^{t} D_i$, where $D_1 \times \cdots \times D_t$ is the domain of f, and the ith dimension is labeled by each $x \in D_i$. The fact that $h_i(x)$ can be stored in space s means that we can partition the rows along the ith dimension into 2^s classes such that if y and z are in the same class (i.e., $h_i(y) = h_i(z)$) then for any other inputs, $f(\ldots, x_i = y, \ldots) = f(\ldots, x_i = z, \ldots)$. It is clear that if we know the classes to which each x_i belongs, then the value of f is uniquely determined. Thus, because the referee receives $h_i(x_i)$ for all i, he can compute $f(x_1, \ldots, x_t)$.

(3) Let g be the (exact) Hamming distance between $t = 2$ sequences, A and B, of n bits each. (This argument can be generalized to $t > 2$.) An item of a stream is a pair (i, v), denoting that v is the value of the ith bit in the corresponding sequence; an adversary controls the order of the pairs. In the merged streams model, we use an array $r[1..n]$ of bits, initialized to zero. Upon receiving (i, v) in A or in B, we set $r[i] = r[i] \oplus v$. We compute $\sum_{i=1}^{n} r[i]$ in a post-processing step, and output the result. Thus $DM_t(g) \leq n$.

We now show that $DD_t(g) \geq t \cdot n$. If we use $< 2n$ space, then at least one of the entities (say Alice) uses space less than n bits. Because there are 2^n different bit sequences of length n, there exist two distinct sequences, say x and y, for which Alice has the same workspace contents and hence sends the same message to the referee. Consider two instances of the problem, one in which $A = B = x$ and the other in which $A = y$ and $B = x$. The messages sent by both Alice and Bob to the referee are the same in both cases and the referee will return the same value, which is a contradiction. ∎

An open problem is to determine the complexity of merged streams vs. distributed streams for randomized algorithms. Kremer et al. [KNR99] presented a proof that for any boolean function, the randomized simultaneous communication complexity with public coins is linear in the sum of the two one-way (Alice-to-Bob and

Bob-to-Alice) communication complexities. However, we recently uncovered a flaw in their proof.[4] Thus it is an open problem to determine the relationship between one-way and simultaneous public coin communication complexity for randomized algorithms. Because of the connections between (a) merged streams and one-way communication and (b) distributed streams and simultaneous communication, any result relating communication models will likely lead to a similar result for the streams models, and perhaps vice-versa.

6 Conclusions

This paper presented a simple technique, coordinated 1-sampling, as an alternative to various probabilistic counting methods previously studied. We showed that this technique is a useful tool for estimating various functions on data streams of interest in practice, enabling improved space and/or time bounds over previous sampling and probabilistic counting approaches. We motivated and formulated the distributed streams model, and presented tight bounds comparing it to previously studied streams models. Two related open problems are the relationship between the randomized complexity for (1) distributed and merged streams, and for (2) simultaneous and one-way communication. As a starting point for the former, it would be interesting to see if the kth frequency moment ($k \geq 3$) algorithm of [AMS96] could be adapted to the distributed streams model.

Acknowledgments. The second author would like to acknowledge Aravind Srinivasan and Eli Upfal for helpful discussions.

References

[AGMS99] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking algorithms for join and self-join sizes. In *Proc. 18th ACM Symp. on Principles of Database Systems*, pages 1–11, May 1999. Full version to appear in JCSS special issue for PODS'99.

[AGPR99a] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 574–576, June 1999. Demo paper.

[AGPR99b] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 275–286, June 1999.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996. Full version to appear in JCSS special issue for STOC'96.

[BCFM98] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. 30th ACM Symp. on the Theory of Computing*, pages 327–336, May 1998. Full version to appear in JCSS special issue for STOC'98.

[CCMN00] M. Charlikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. 19th ACM Symp. on Principles of Database Systems*, pages 268–279, May 2000.

[CMN98] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 436–447, June 1998.

[4] We reported the flaw to the authors, who agreed with us [NR00]. Specifically, the following is a counter-example to the argument in the last paragraph of the proof of Theorem 5.1 (Theorem 10 in the STOC version) that α_0 is small: We assume the reader is familiar with the terminology of their proof. In each rectangle, we place all the weight μ equally on two points that do not share a row or column. On one point the function is 1, and the other point the function is 0. The one-way protocols correctly compute these two points (i.e., even though Alice passes Bob the same message, Bob can correctly distinguish based on his input, and vice-versa). The rectangle is good, the columns with the weight are good, the rows with the weight are good, but $\alpha_0 = 1/2$. Thus the overall error in the simultaneous protocol is not small, contradicting the main claim of the proof.

[FKSV99a] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, pages 501–511, October 1999.

[FKSV99b] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. Technical report, AT&T Shannon Laboratories, Florham Park, NJ, July 1999.

[FM85] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Computer and System Sciences*, 31:182–209, 1985.

[FS00] J. Fong and M. Strauss. An approximate L^p-difference algorithm for massive data streams. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science, LNCS 1770*, pages 193–204. Springer, February 2000.

[GM98] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[GM99] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In J. M. Abello and J. S. Vitter, editors, *External Memory Algorithms*, pages 39–70. AMS, 1999. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 50. Also short paper in SODA'99.

[GMMO00] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, November 2000.

[HNSS95] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. 21st International Conf. on Very Large Data Bases*, pages 311–322, September 1995.

[HRR98] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Digital Systems Research Center, Palo Alto, CA, May 1998.

[IKM00] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series datasets using sketches. In *Proc. 26th International Conf. on Very Large Databases*, pages 363–372, September 2000.

[Ind00] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, November 2000.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, UK, 1997.

[KNR99] I. Kremer, N. Nisan, and D. Ron. On randomized one-round commmunication complexity. *Computational Complexity*, 8(1):21–49, 1999. Preliminary version in STOC'95.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[New91] I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39:67–71, 1991.

[NR00] N. Nisan and D. Ron. Private communication, October-November 2000.

[NS96] I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 561–570, May 1996.

[TPC00] Transaction processing performance council (TPC). *TPC Benchmarks*, 2000. URL: `www.tpc.org`.

A Appendix

A.1 Proof of the time complexity in Theorem 2

We first analyze the time taken to process a bit by an instance, A_k, of the algorithm at Alice. For each bit a_i, A_k does the following: (1) Compute $e_k(i)$. (2) If there is no space to store the item, then move to a higher level and evict those that do not belong at the higher level.

We store the sample S_k as an array $S_k[1 \ldots d]$ of linked lists, with $S_k[i]$ containing all those positions j for which $e_k(j) = i$, i.e., those items that "die" at level i. Upon receiving a 1-bit whose position has been selected for S_k, the position is inserted into the appropriate list. When the algorithm needs to move to the next level, it simply deletes all the positions that are in the list corresponding to the current level. Because every position is inserted and deleted at most once from this data structure, the total work done over all the n input items is just $O(n)$ plus the time for the computation of $e_k(j)$, for each 1-bit in $j = 1, \ldots, n$. If we assume that we can multiply and add two $\log n$ bit numbers in $GF(n)$ in constant time, then this would give us an amortized constant time per bit for A_k. Hence the time complexity of this algorithm is $O(s_2)$ amortized, which is $O(\log(1/\delta))$.

If we use lazy discarding when changing levels, so that once $|S_k|$ reaches $c\alpha$, it remains there, the amortized bound becomes a worst-case expected bound. (It becomes an expected bound due to the time needed to find the next level with items to delete, once a current level is exhausted.)

A.2 Proof of Lemma 3

We will need the following additional lemmas.

Lemma 6 *For a given l, the random variables $\{X_{li} | 1 \leq i \leq n\}$ are pairwise independent.*

Proof. For distinct i, j,

$$\begin{aligned}&\mathbf{Pr}\{(X_{li} = 1) \wedge (X_{lj} = 1)\} = \mathbf{Pr}\{(e(i) \geq l) \wedge (e(j) \geq l)\} \\ &= \mathbf{Pr}\{e(i) \geq l\} \cdot \mathbf{Pr}\{e(j) \geq l\} = \mathbf{Pr}\{X_{li} = 1\} \cdot \mathbf{Pr}\{X_{lj} = 1\}\end{aligned}$$

because $e(i)$ and $e(j)$ are pairwise independent. ∎

Lemma 7 *For $l \in \{0 \ldots d-1\}$, $E[X_l]$ is $\mu_l = \frac{U}{2^l}$ and the variance of X_l is $\sigma_l^2 = \frac{U}{2^l}(1 - \frac{1}{2^l})$*

Proof.

$$E[X_l] = E[\sum_{i \in 1_U} X_{li}] = \sum_{i \in 1_U} E[X_{li}] = |1_U| \cdot E[X_{li}]$$

where we have used linearity of expectation. $E[X_{li}]$ is the probability that i made it to level l, which is $\mathbf{Pr}\{e_k(i) \geq l\}$, i.e., $\frac{1}{2^l}$. Since $|1_U|$ is U, we have $E[X_l] = \frac{U}{2^l}$.

Because the X_{li}'s are pairwise independent (Lemma 6), we have the following from [MR95], pp.52:

$$\sigma_l^2 = \sum_{i \in 1_U} \sigma_{X_{li}}^2 = |1_U| \cdot \sigma_{X_{li}}^2 = \frac{U}{2^l}\left(1 - \frac{1}{2^l}\right)$$

∎

We are now ready to prove Lemma 3.

Proof. If $U \leq c\alpha$, then the "sketches" stored at Alice and Bob would contain all the 1-bits in the input, and hence the algorithm would compute U exactly. So assume that $U > c\alpha$.

Let μ_k and σ_k denote the mean and standard deviation of X_k respectively.

$$\mathbf{Pr}\{B_k\} = \mathbf{Pr}\{|X_k - \mu_k| > \epsilon\mu_k\}$$

Using Chebyshev's inequality, $\mathbf{Pr}\{|X_k - \mu_k| \geq t\sigma_k\} \leq \frac{1}{t^2}$ and substituting $t = \frac{\epsilon\mu_k}{\sigma_k}$, we get:

$$\mathbf{Pr}\{B_k\} \leq \frac{\sigma_k^2}{\epsilon^2 \mu_k^2} \tag{1}$$

Substituting the values of μ_k and σ_k^2 from Lemma 7 into equation (1), we get

$$\mathbf{Pr}\{B_k\} \leq \frac{2^k}{U\epsilon^2}\left(1 - \frac{1}{2^k}\right) < \frac{2^k}{U\epsilon^2}$$

Thus,

$$\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} \quad < \quad \sum_{i=0}^{l} \frac{2^k}{U\epsilon^2} = \frac{2^{l+1}-1}{U\epsilon^2} < \frac{2^{l+1}}{U\epsilon^2}$$

Because l is the first level such that $\frac{U}{2^l} < C\alpha$, we have $\frac{U}{2^{l-1}} \geq C\alpha$, i.e., $\frac{U}{2^l} \geq \frac{C}{2\epsilon^2}$. (Recall that $U > c\alpha > C\alpha$.) Thus,

$$\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < \frac{4}{C}$$

We choose $C = 36$ and Lemma 3 is proved. ∎

A.3 Proof of Lemma 4

Proof. The sum $\sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\}$ is the probability that the algorithm stops in level $l+1$ or greater (because the S_i's are mutually exclusive). This is exactly the probability that at level l, we still have more than $c\alpha$ 1's, i.e., the probability that $X_l > c\alpha$.

$$\begin{aligned}
\mathbf{Pr}\{X_l > c\alpha\} &= \mathbf{Pr}\{X_l - \mu_l > c\alpha - \mu_l\} \\
&\leq \mathbf{Pr}\{X_l - \mu_l > c\alpha - C\alpha\} \quad \text{because} \quad \mu_l < C\alpha \\
&\leq \frac{\sigma_l^2}{(c-C)^2\alpha^2} \quad \text{using Chebyshev's inequality}
\end{aligned}$$

From Lemma 7 we have $\sigma_l^2 = \frac{U}{2^l}(1 - \frac{1}{2^l}) < \frac{U}{2^l}$. Since $E[X_l] < C\alpha$, we have $\frac{U}{2^l} < C\alpha$. Using this in the above expression,

$$\begin{aligned}
\mathbf{Pr}\{X_l > c\alpha\} &< \frac{C}{(c-C)^2\alpha} = \frac{C\epsilon^2}{(c-C)^2} \\
&= \frac{\epsilon^2}{36} \quad \text{by choosing} \quad c = 72, C = 36 \\
&< \frac{1}{9} \quad \text{since} \quad \epsilon \leq 1
\end{aligned}$$

∎