

Finding Global Icebergs over Distributed Data Sets *

Qi (George) Zhao
College of Computing
Georgia Tech

Mitsunori Ogihara
Computer Science Dept.
Univ. of Rochester

Haixun Wang
IBM T.J. Watson
Research Center

Jun (Jim) Xu
College of Computing
Georgia Tech

ABSTRACT

Finding icebergs – items whose frequency of occurrence is above a certain threshold – is an important problem with a wide range of applications. Most of the existing work focuses on iceberg queries at a single node. However, in many real-life applications, data sets are distributed across a large number of nodes. Two naïve approaches might be considered. In the first, each node ships its entire data set to a central server, and the central server uses single-node algorithms to find icebergs. But it may incur prohibitive communication overhead. In the second, each node submits local icebergs, and the central server combines local icebergs to find global icebergs. But it may fail because in many important applications, globally frequent items may not be frequent at any node. In this work, we propose two novel schemes that provide accurate and efficient solutions to this problem: a sampling-based scheme and a counting-sketch-based scheme. In particular, the latter scheme incurs a communication cost at least an order of magnitude smaller than the naïve scheme of shipping all data, yet is able to achieve very high accuracy. Through rigorous theoretical and experimental analysis we establish the statistical properties of our proposed algorithms, including their accuracy bounds.

Categories and Subject Descriptors

E.1 [DATA STRUCTURES]; F.2 [Theory of Computation]: Analysis of Algorithm and Problem Complexity

General Terms

Algorithms, Measurement, Theory

Keywords

Data Streaming, Icebergs, Statistical Inference

*The work of Qi Zhao and Jun Xu is supported in part by NSF grant NETS-NBD 0519745 and NSF CAREER Award ANI 0238315.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'06, June 26–28, 2006, Chicago, Illinois, USA.
Copyright 2006 ACM 1-59593-318-2/06/0003 ...\$5.00.

1. INTRODUCTION

Finding items whose frequency of occurrence is greater than a certain threshold is a well-explored problem [10, 20]. Most of the existing work focuses on iceberg queries at a single node [5, 18, 22, 26]. However, in many real-life applications, data sets are physically distributed over a large number of nodes. We study the problem of finding items whose frequencies of occurrences across these nodes add up to exceed a certain threshold, i.e., finding global icebergs over aggregate data.

Identifying global icebergs has applications in wide range of areas ranging from network monitoring to biosurveillance. For example, applications that detect DDoS attacks [25] try to find destination IP addresses (victims of a DDoS attack) that occur frequently in IP traffic aggregated over many network ingress points. Since the attacking packets may come from a large number of hosts (called Zombie hosts) and may traverse many different Internet paths, an individual ISP (Internet service provider) may not see a large number of packets destined to any victim. In other words, such global icebergs (IP addresses that are targets of the DDoS attacks) may not emerge as local icebergs anywhere, and therefore may not be detectable without correlating data from multiple monitoring points. Another network monitoring application [25] is that of finding frequently accessed objects/URLs (icebergs) in a Content Delivery Network (CDN) that contains many CDN nodes (e.g., Akamai [1] has thousands of nodes). Each object/URL can be cached and served by multiple CDN nodes. To find the globally frequently accessed objects/URLs, we must monitor and correlate the frequency counts on all CDN nodes. Global iceberg query also has applications in system monitoring. It is suggested to us [28] that finding DLLs (Dynamically Linked Libraries) that have been modified on a large number of hosts inside an organization may help detect the spread of an unknown worm or spyware. Recently, the possibility of bioterrorist acts has highlighted the need of biosurveillance, which monitors a large population for changes against a predetermined norm. To achieve this goal, we need effective linking of data from a large number of different sources so that we can catch subtle local changes that are emerging globally.

Two naïve approaches might be considered for finding global icebergs. First, we may ship the data from all the nodes to a central server, which takes their aggregation and then applies the algorithms which are designed for detection of icebergs on a single node [5, 18, 22, 26]. This method is clearly inefficient and often infeasible due to the cost of communication between the server and the nodes, when there are large number of nodes with a large amount of data on them.

In the second approach, each node applies local iceberg algorithms on its own data set to find local icebergs, and the central server aggregates the local icebergs to find the global icebergs. This

approach reduces communication overhead, but unfortunately, it may miss items which are infrequent in local nodes, but their aggregated frequencies across all nodes exceed the threshold of global icebergs. For instance, assume on a particular day a URL is accessed 10,000 times globally, which qualifies it as a global iceberg. However, these 10,000 accesses come from 10,000 different ISPs, which means the URL is not a local iceberg in any ISP.

Nevertheless, the central server relies on data or messages from distributed nodes to find the global icebergs. The naïve solutions above showed that messages that contain the entire local data sets are inefficient, and messages that contain only the local icebergs are lossy. Thus, the core problem of finding global icebergs is to design an appropriate scheme for summarizing local data. In other words, each node should send a small yet informative message to the server so that the server can derive global icebergs in an efficient manner. In this paper, we propose two schemes for this purpose.

1.1 Problem Statement

We formulate the problem as follows. Consider a system or network that consists of N distributed nodes. The data set S_i at node i contains a list of $\langle \text{item}, \text{count} \rangle$ pairs, corresponding to items and their exact frequency counts in the local dataset. We want to find the set of items whose total counts across all the nodes add up to exceed a certain threshold T .

In addition, instead of making a binary decision of whether an item is an iceberg or not, we want to provide accurate estimates of the total counts for all the potential icebergs. In other words, we want to find item x along with its estimated total count c , such that $c \geq (1 - \epsilon)T$, where ϵ is a small constant (say 0.1). Thus, we are tackling an estimation problem. Although it is strictly harder than traditional iceberg problems that require binary decisions, we still refer to it as the “global iceberg problem”.

We are interested in solutions for the above problem that require a minimum amount of communication between these nodes and the central monitoring server. In particular, we are interested in one-way¹ (unidirectional) protocols, in which each node only needs to send a significantly reduced amount of data to the central server, and the server does not need to send any message back to the local nodes (except perhaps for TCP ACK’s) and/or request additional data. Conceptually, each message from a node contains a highly space-efficient encoding of the frequency counts of the items at the node. The server will decode messages from all of the nodes to find the global icebergs. The challenge is to find the right encoding and decoding schemes.

Note that we distinguish between two different types of global iceberg problems, namely, finding icebergs over *distributed bags* and over *distributed streams* [22]. In both types, we assume that each node will process a stream of data items. The difference is that, in the case of *distributed bags*, each node has enough memory and computation power to “digest” its local stream with zero information loss. In other words, after processing the stream, each node obtains a list of items and their exact frequency counts in the local stream. In the case of iceberg query over *distributed streams*, completely accurate processing is not possible for high-speed data streams because of memory and computation constraints. In this case, we may have to find icebergs over multiple streams where each of these streams is processed with some information loss concerning the frequencies of items. Clearly, it is in general more

¹Note that the efficiency and accuracy of the protocol may be improved if multiple back-and-forth data exchanges are allowed (i.e., more than one-way). However, our preliminary study finds that one-way communication can be made very accurate and efficient, thus the benefit of additional interaction is not immediately clear.

challenging to find icebergs over distributed streams than over distributed bags. In this work we solve for the bag case², leaving the stream case for future work.

1.2 Our Solutions and Contributions

We propose two solutions to the global iceberg problem: a sampling based approach and a counting sketch based approach. In our sampling approach, each node samples a list of data items along with their frequency counts in the local data bag and send them to the server, which for each distinct item, aggregates its sampled frequency counts scaled by the inverse of their respective sampling rates to obtain an estimate of its total frequency. An obvious question is “what is a good sampling strategy?”. More specifically, consider that at a node, one item A has count 1 and another item B has count 100. Should we sample the former with the same probability as the latter? Intuitively, the answer is “no” because the count of B has bigger potential to help push the aggregate count of B over the threshold T than that of A . Therefore we should sample the pair $\langle B, 100 \rangle$ with higher probability than $\langle A, 1 \rangle$, and more generally, the probability with which a node samples an item should be an increasing function g of its frequency count c . But what kind of function should this $g(c)$ be? We answer this question in a comprehensive way. We obtain a near-optimal sampling strategy that results in almost the lowest possible estimation error under certain resource (communication cost) constraints. We also uncover the fundamental mathematical structures behind the global iceberg problem (over flat infrastructure). In addition, we develop a new large deviation theorem that is not only critical for establishing a tight accuracy bounds in both our sampling and sketch based solutions, but may also be useful in other distributed data streaming applications.

Our second solution, the sketch-based approach, detects icebergs in a much more communication-efficient way than the sampling-based solution³. It is inspired by the observation that the sampling process in our sampling-based approach implicitly serves two purposes simultaneously. The first purpose is to obtain the identities of the items that may potentially become icebergs. The second purpose is to use it as a counting device, encoding the frequency counts of items (subject to information loss due to sampling). We observe that although sampling is great for gathering identities of the items, it is not a great counting device. In fact, a counting sketch, such as the one in [24], may be able to encode the frequency counts of various items in a much more resource-efficient way than sampling. Based on this observation, our approach works as follows. Each node will not only summarize its data into a counting sketch, but also sample a small⁴ percentage of identities of the items, and send both to the server. For each sampled identity, the server will use it to query all the collected counting sketches and add up the approximate counts to obtain an estimate of the total frequency count of the item.

One challenge we face in this approach is to find the right count-

²The solution proposed by Manjhi et al. [25] works for the stream case. However, their solution is for the hierarchical communication infrastructure and will work well for neither bag nor stream case on flat infrastructure, which our solutions target. We will discuss this in detail in Section 2.

³The presentation of our sampling-based solution, however, is necessary since the near-optimal sampling strategy and the underlying mathematical structure are not only new discoveries but also implicitly used in our sketch-based solution.

⁴Compared to the pure sampling approach, the sampling rate here can be much smaller, since we only need to ensure that each item that has large overall frequency count has *at least one of its occurrences sampled*.

ing sketch. As we will discuss later, existing counting sketches such as Space-Code Bloom Filter [24], Spectral Bloom Filter [13] and CM sketch [16] are not well suited for this application. We design a novel counting sketch that is especially accurate and efficient for detecting icebergs, based on insights into the mathematical structure of the global iceberg learned in designing our sampling-based approach.

Through rigorous mathematical analysis, we show that both of our solutions achieve high estimation accuracy with a communication cost at least an order of magnitude smaller than the naive approach of shipping all data to the server. In addition, we evaluated the performance of both solutions on two real-world data sets obtained from Abilene network and IBM. Experimental results confirm the high accuracy and efficiency of both solutions predicted by the analysis.

The rest of this paper is organized as follows. We start with a discussion of the related work in Section 2. Then we describe the design of our sampling-based scheme and develop the theory of good sampling strategies, in Section 3. Next we present a more accurate scheme, i.e., the counting-sketch-based scheme and its theoretical analysis in Section 4. Section 5 evaluates the sketch-based scheme using real-world data sets. We conclude our work in Section 6.

2. RELATED WORK

The term *iceberg queries* was first introduced to describe database queries that find records whose aggregate values over an attribute are above a certain threshold [20]. Many applications compute a simple type of iceberg query — identifying in a multiset items with frequency above a certain threshold.

Traditional iceberg queries are computed over data bags. For example, a number of heuristics proposed by Fang *et al.* [20] make multiple scans of the data. Recently, finding icebergs over streaming data has attracted much research effort [9, 5, 18, 22, 15, 9, 26]. Among them, Karp *et al.* [22] gave the space and time bound for computing exact iceberg queries for both streams and bags. Charikar *et al.* [9] developed a randomized algorithm to identify frequent items in a stream. Cormode *et al.* [15] found frequent items in a dataset that is undergoing deletion operations as well as insertions.

All the above work focuses on finding icebergs at a single node, and the central issue is how to design a synopsis data structure called counting sketches to tradeoff accuracy against space and time. One may wonder that since almost all of these counting sketches are linearly composable in the sense that the counting sketch of the overall data set $\bigcup_{i=1}^N S_i$ can be computed as the aggregate of the counting sketches of the individual data sets S_i , $i = 1, 2, \dots, N$, some of them may provide a better solution to the global iceberg problem than our approach. This is not the case for two reasons. The first reason is that these counting sketches typically allocate similar amount of storage/communication resource to an item of large or small count alike, which may not be resource-efficient for the purpose of detecting global icebergs. Our approach, on the contrary, allocates the right amount of storage/communication resource to an item according to its potential contribution to push its total count over the threshold. The second reason is that the accuracy guaranty in the counting sketches is typically in the form of ϵ times the L_1 norm of whole data stream (i.e., the sum of all counts of all items). In our approach, the accuracy guarantee is in the form of ϵ times the total count of an item across all the nodes. In the technical report version of the paper [29], we compare our approach with the CM sketch [16], a representative counting sketch, with respect to accuracy and communication overhead, and show that our approach requires much less commu-

nication overhead than the CM sketch to achieve the same accuracy in detecting global icebergs. It is omitted here due to interest of space.

Distributed streaming has recently been studied in the theoretical computer science and database contexts [21, 6, 11]. The iceberg problem is first introduced to the distributed environment in [25]. Manjhi *et al.* [25] proposed a method to find icebergs in the union of multiple distributed data streams. It arranges nodes in a multi-level communication structure. Each node uses traditional synopsis data structures (e.g., [26]) to find “local icebergs,” and then sends the synopsis to its parent in the communication hierarchy. The paper shows that, by giving each node an error tolerance according to its level in the hierarchy, the approximate synopses computed at lower levels can be combined to recover icebergs at a higher level within certain accuracy. This algorithm, however, does not solve our problem of finding icebergs using a one-level (flat) communication infrastructure.

By combining local icebergs to find global icebergs, the above approach essentially assumes that a globally frequent item is also frequent locally somewhere. This assumption works well if the global iceberg threshold is very high relative to the number of distributed nodes. However, when the number of nodes is very large, a globally frequent item may not be frequent in any local site⁵. For example, if we cut up an iceberg of size 10,000 and distribute it evenly across 10,000 nodes, the algorithm of Manjhi *et al.* [25] will not be able to identify this iceberg unless the leaf nodes send all their data to their parents⁶. In our problem, where a one-level (flat) communication infrastructure is assumed, this is translated into every node sending all its data to the server, which is unacceptable. Therefore, the approach of Manjhi *et al.* [25] does not offer an efficient solution to our problem even for the bag case. A key contribution of our work is a novel counting sketch that is specially optimized for finding global icebergs.

Another problem related to answering iceberg queries is to find the most frequent items in data streams, known as top-K. Techniques for answering top-K queries have been proposed for a single stream as well as a distributed set of data sets (bags) [10, 7, 27, 17]. Babcock *et al.* [7] studied the problem of monitoring top- k items in a distributed environment. The work is extended by Olston *et al.* [27] to support both sum and average queries on data spread over multiple sources. The work is extended also by Das *et al.* [17] to support estimation of set-expression cardinality in a distributed streaming environment. These approaches all try to keep local top-K items at each node in alignment with the global top-K items. In Babcock *et al.*’s approach [7], for instance, each local site is assigned a factor that represents an allowed local skew, and the sum of the local skew is kept within the global error tolerance. As long as the local skew is within the assigned factor, the global top-K will not change and no communication is required between local sites and the central server. In this sense, the techniques used in the above approaches for solving the top-K problem are similar to the techniques used by Manjhi *et al.* [25] for the iceberg problem. Techniques for solving top-K queries generally cannot be directly used for solving iceberg queries since the size of an iceberg can be much smaller than top-K items, making it much harder to find.

⁵Many real-life applications fall into this category.

⁶This may be acceptable in Manjhi *et al.* [25] due to the distributed nature of the hierarchical communication infrastructure and the fact that much less data need to be transferred when the level becomes higher.

3. THE SAMPLING-BASED SCHEME

An obvious approach to find global icebergs over distributed bags is sampling. In this approach, each item-count pair $\langle I, c \rangle$ at a node is sampled with some probability and the list of sampled pairs are sent to the server; after collecting all the samples from all nodes, the server estimates the total count of each item that appears at least once in the aggregate list by aggregating the counts in the samples and compensating for sampling. We state earlier that the probability with which a node samples an item in general should be an increasing function g of its frequency count c . But what kind of function should this $g(c)$ be?

To answer this question, we need to come up with a mathematically sound measure to compare different sampling strategies, determined by their choices of the sampling function g , and then we pick the one that optimizes this measure. We show that one such measure is the *worst-case* mean square error (MSE). Let \hat{X} be the estimator for a random variable X . Then its MSE is defined as $E[(\hat{X} - X)^2]$. The notion of minimizing the worst-case MSE can be explained by the following adversarial minimax model. Imagine that an adversary tries to split an iceberg into small pieces and hide it among many nodes. Given our choice of g and the corresponding estimator, the adversary would choose the split pattern that maximizes the MSE of our estimator. The only thing we can do is to minimize this maximum (worst-case) MSE, since in general neither the nodes nor the server have a control over the split pattern.

In the following, we will describe our sampling process (Section 3.1), characterize the type of g we will use that attempts to minimize the aforementioned worst-case MSE (Section 3.2), and then establish tight tail bounds of the estimation based on an improved large deviation theorem (Section 3.3) developed by us.

3.1 Sampling Process and the Estimator

Recall that there are total N nodes in the system. Suppose that the frequency count of an item I at a node i is $c_i \geq 0$, $i = 1, \dots, N$. Our goal is to estimate $f_I = \sum_{i=1}^N c_i$. Due to sampling only a part of these (c_i) 's may appear in the aggregate list at the server. Suppose that the server received k of them, denoted as x_1, \dots, x_k , from nodes j_1, j_2, \dots, j_k respectively. In other words, $x_i = c_{j_i}$ for $1 \leq i \leq k$.

Since each count c is independently selected with probability $g(c)$, for each occurrence of c , on the average $\frac{1}{g(c)} - 1$ other nodes have $\langle I, c \rangle$ but did not sample it; that is, each occurrence of c can be thought of as representing $\frac{1}{g(c)}$ occurrences of c . Thus, we estimate the total count f_I of I as

$$\hat{f}_I = \frac{x_1}{g(x_1)} + \frac{x_2}{g(x_2)} + \dots + \frac{x_k}{g(x_k)} \quad (1)$$

The following theorem shows that this estimator is unbiased and establishes its MSE.

THEOREM 1. *The estimator \hat{f}_I is unbiased, and its MSE is $\sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$.*

PROOF. Consider the random variables y_i , $1 \leq i \leq N$:

$$y_i = \begin{cases} c_i & \text{with probability } g(c_i), \\ 0 & \text{with probability } 1 - g(c_i). \end{cases}$$

Then the estimator in (1) is rewritten as $\sum_{i=1}^N \frac{y_i}{g(c_i)}$, by treating $\frac{0}{0}$ as 0 (for (c_i) 's that is equal to 0). For all i , $1 \leq i \leq N$, $E[y_i] = c_i g(c_i)$ so $E[\frac{y_i}{g(c_i)}] = \frac{E[y_i]}{g(c_i)} = c_i$. This means that the estimator is unbiased, and therefore, the MSE of f_I is equal

to its variance: $\text{Var}[f_I] = \text{Var}[\sum_{i=1}^N \frac{y_i}{g(c_i)}] = \sum_{i=1}^N \frac{\text{Var}[y_i]}{g(c_i)^2} = \sum_{i=1}^N \frac{c_i^2 g(c_i)(1-g(c_i))}{g(c_i)^2} = \sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$. \square

3.2 The Worst-Case MSE and the Near-Optimal Sampling Strategy

We obtain from Theorem 1 that the MSE of f_I is $\sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$. Its value clearly depends on how the quantity f_I is split into c_1, \dots, c_N . As mentioned above, for each fixed function g , we think of the worst-case split of f_I by an adversary, which is the one that maximizes the MSE with respect to g . We would like to choose g that minimizes this worst-case MSE. Note that we can always reduce the worst-case MSE by increasing the sampling rates (i.e., making $g(c)$ larger for all c values), albeit at higher communication cost. Therefore, g should be constrained to a family \mathcal{G} that has the same communication cost (elaborated next). In other words, we wish to solve the optimization problem:

$$\min_{g \in \mathcal{G}} \max_{c_1, \dots, c_N: c_1 + \dots + c_N = f_I} \sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)} \quad (2)$$

The minimax problem in Formula (2) is in fact not as well-formed as it looks, as most of the complications are hidden in the term " $g \in \mathcal{G}$ ". We intend to choose g from a family \mathcal{G} that has the same communication cost. However, it is very hard (if not impossible) to come up with a right measure of the communication cost that both is mathematically sound and allows the above minimax problem to be solved exactly. An obvious and mathematically sound measure is the expected total size of the sampled items. For example, suppose there are n items of sizes s_1, s_2, \dots, s_n in a bag. Then the average communication cost under this measure is $\sum_{i=1}^n g(s_i)$. However, this measure does not lead to a clean solution since the communication cost is a function of the sizes of all the items at all the nodes, which should not be a part of the equation since it is not known when a node applies sampling. Therefore, we would like the communication cost to be defined as a function of just g and the split pattern. However, the problem with definitions like this is that they are often at odds with our intuitions of "communication cost", making the resulting mathematical solution (if any) practically unappealing (no matter how clean it is).

Our approach is to stick with the above intuitive definition of the communication cost, but to solve for a fairly good solution instead of the optimal one. Our idea is to design our sampling strategy such that the MSE is independent of the split pattern, i.e., each split pattern is equally bad (or equally good) as any other split patterns. In this case, every case is equivalent to the worst-case. This completely eliminates the power of the adversary to increase the MSE by manipulating the split pattern, although there is some room for it to impose a slightly larger communication cost. We assume that the adversary only has control over the split patterns of its own few items. Therefore, the increase of the overall communication cost is usually negligible. The remaining question is how this independence between the MSE and split pattern can be enforced. We have the following exact answer to this question.

PROPOSITION 1. *The independence condition is satisfied if and only if $g(c) = \frac{c}{d+c}$ for some constant $d \geq 0$.*

PROOF. Let us first solve for the "if" part. For each i , $1 \leq i \leq N$, $\frac{c_i^2(1-g(c_i))}{g(c_i)}$ is equal to $d c_i$, then the MSE is $d \sum_{i=1}^N c_i = d f_I$ since $f_I = c_1 + \dots + c_N$, and the condition is satisfied. For the "only if" part, let us first consider the split pattern in which all the "mass" f_I is concentrated on one node. The MSE of our estimator with this split pattern is $F(f_I)$, where $F(c)$ denotes the function

$\frac{c^2(1-g(c))}{g(c)}$. Then we consider another split pattern in which the mass is evenly distributed over f_I different nodes. The MSE in this case is $f_I F(1)$. According to the independence condition, we have $F(f_I) = f_I F(1)$. Since this relation holds for arbitrary f_I values, it can be shown that the function $F(x)$ has the form dx for any positive integer value x , where $d = F(1)$. By resolving $\frac{c^2(1-g(c_i))}{g(c_i)} = dc_i$, we obtain $g(c_i) = \frac{c_i}{d+c_i}$. \square

3.3 Tight Tail Bounds Through a New Large Deviation Theorem

So far we know the variance of our estimator, i.e., $\text{Var}[\hat{f}_I]$ is equal to df_I no matter how f_I is split. We now wish to obtain a tight tail bound on the estimator; that is, a tight upper bound on $\Pr[|\hat{f}_I - f_I| > b]$, the probability that the estimation error has deviation larger than b . Since we know the MSE of \hat{f}_I , a standard tail bound can be obtained using Chebyshev's inequality, but it offers only a loose bound since it does not take into consideration that \hat{f}_I is the sum of independent random variables $y_1/g(c_1), \dots, y_N/g(c_N)$ (defined in the proof of Theorem 1). A much tighter bound can be obtained using Chernoff-Hoeffding's inequality [12], but it is not ideal since it does not take advantage of the fact that the estimator has small variance.

We establish the following tail bound theorem that takes advantage of both the independence and the small variance. This is a one-sided bound and a similar two-sided bound can be easily obtained (with the RHS doubled). This theorem improves Theorem A.1.19 in [4, pp.270] by sharpening ϵ to $\frac{\epsilon}{3}$ on the RHS of the inequality. We are able to obtain tail bounds of $\hat{f}_I - f_I$ from this Theorem that are much tighter than obtainable from Chernoff-Hoeffding's inequality [12]. Its proof can be found in [29]. This theorem will also be used to establish tail bounds in our counting sketch based scheme.

THEOREM 2. *For every $\theta > 0$ and $\epsilon > 0$, the following holds: Let $W_i, 1 \leq i \leq m$, m arbitrary, be independent random variables with $E[W_i] = 0$, $|W_i| \leq \theta$ and $\text{Var}[W_i] = \sigma_i^2$. Let $W = \sum_{i=1}^m W_i$ and $\sigma^2 = \sum_{i=1}^m \sigma_i^2$ so that $\text{Var}[W] = \sigma^2$. Let $\delta = \ln(1 + \epsilon)/\theta$. Then for $0 < a \leq \delta\sigma$,*

$$\Pr[W > a\sigma] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$$

We now map this large deviation theorem to our scheme. The error of our estimator (a random variable) corresponds to $W (= \hat{f}_I - f_I)$. The error in our estimation of each c_i ($y_i/g(c_i)$ where y_i is defined in the proof of Theorem 1) corresponds to W_i , i.e., $W_i = y_i/g(c_i) - c_i$ for all c_i . The MSE/variance of our (unbiased) estimator corresponds to σ^2 in Theorem 2. Theorem 2 essentially states that the probability that the estimation error W is larger than a times standard deviation is no more than $e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$. However, this is true only for $a \leq \delta\sigma$, and δ, ϵ and θ are related by $\delta = \ln(1 + \epsilon)/\theta$. Therefore, we need to bound θ , which is the upper bound for $|W_i|, i = 1, 2, \dots, n$, since otherwise either ϵ has to be large or a has to be very small, and both cases lead to trivial or loose tail bounds. This is achieved by setting a cutoff point for sampling as follows.

We know that for all⁷ c_i equal to 0, $W_i = 0 - 0 = 0$, and for all $c_i > 0$, $W_i = c_i/\frac{c_i}{c_i+d} - c_i = d$ if c_i is sampled, and $W_i = 0 - c_i = -c_i$ otherwise. Therefore, $|W_i| \leq \max\{d, c_i\}$. So if we do not set a cutoff point, θ has to be larger than or equal to $\max\{d, \max\{c_1, c_2, \dots, c_N\}\}$ which may be a very large

⁷Here we treat $\frac{0}{0}$ as 0 by convention.

value considering some big value of c_i and hence lead to loose tail bounds. Instead we set a cutoff point M in the sense that for any item-count pair $\langle I, c \rangle$, when $c > M$, we sample it with probability 1 instead of $g(c)$. These pairs would not introduce any error to our estimation. So we only need to consider the left item-count pairs. In this case θ can be set to $\max\{d, M\}$. The remaining question is what is the value of M . We set it to d . The reason is that if we set M less than d , there is no improvement on θ (and hence the bound) because $\theta \geq d$, but if we set M larger than d , θ becomes larger than d , resulting in worse bound. Notice that an item-count pair whose count is greater than d will have at least 1/2 chance of being sampled even if there is no cutoff point. Therefore, use of cutoff point d will increase the average communication cost of some item-count pairs by at most a factor of two. This should not impact the communication bandwidth too much considering the small number of items whose (local) frequencies are larger than d in real data.

3.4 Numerical results

In this section, we illustrate the worst-case MSE and tail bound of our estimator using a numerical example. We assume that our system consists of $N = 10,000$ nodes. We compute the numerical results for $f_I = 10,000$ and $f_I = 5,000$. We want to send no more than a small fraction (say 2%) of the data to the server. Recall that an item with count c is sampled with probability $\frac{c}{c+d}$. We set $d = 49$, which translates into a 2% sampling rate for items with count value 1. By Theorem 1, the standard deviation of the estimator is 495 and 700 in the cases of $f_I = 5,000$ and $f_I = 10,000$ respectively. By Theorem 2, we establish that $\Pr[4,030 \leq \hat{f}_I \leq 5,970] \geq 76\%$ when $f_I = 5,000$. In other words, the probability that the error (when $f_I = 5,000$) is within 20% is about 0.76. When $f_I = 10,000$, we get $\Pr[8,647 \leq \hat{f}_I \leq 11,353] \geq 76\%$. In other words, the probability that the error (when $f_I = 10,000$) is within 13% is about 0.76. It shows that the value of d , which determines the sampling rate, crucially affects the tradeoff between communication cost and accuracy.

4. COUNTING-SKETCH-BASED SCHEME

We observe that the sampling-based method is not very accurate because typically, the sampling rate is low due to limited communication bandwidth. More importantly, sampling is not efficient for counting. For example, suppose the adversary cuts up an iceberg of size 10,000 into 10,000 1's and distributes them evenly among 10,000 nodes. With a sampling rate of 10%, on the average 1,000 nodes will send in the identity and the frequency (which is 1) of this item to the server. The identity of the item, which can be hundreds of bits long in many applications, will be repeated for approximately 1,000 times. Such repetitions result in poor use of communication bandwidth.

These observations lead us to consider the use of counting sketches. A carefully designed counting sketch such as [14, 24] is likely to offer much higher accuracy in estimating counts, but it alone cannot complete the task of finding icebergs. This is because a counting sketch is essentially a query interface, that is, to inquire about the count of an item its identity has to be known first. This means that the server would require a separate list of item identities L for which it will inquire about their global counts in the counting sketches. Our scheme makes each node sample some candidate identities and send them to the server, which the server will merge them into L . Note that the sampling rate here for generating candidate identities can be significantly smaller than that of the sampling-based approach, for we only have to ensure that each item that has a large global count is *sampled at least once*. In contrast, in the sampling-based scheme we need a relatively large

percentage of the item identities sampled to achieve good accuracy in counting.

In this section we design a novel counting sketch that is particularly efficient and accurate for finding global icebergs. We optimize its parameters based on the principle of minimizing worst-case MSE (described earlier in the sampling-based approach). We show that with the same communication overhead as the sampling-based scheme, this scheme has much smaller variance and produces much more accurate estimates. Before designing our own counting sketch, we investigated the possibility of using existing ones such as Spectral Bloom Filter (SBF) [14] and the Space-Code Bloom Filter (SCBF) [24]. We found that SCBF is not very efficient in encoding a bag since it introduces unnecessary noises through a random “balls into bins” process that is necessary for encoding a high-speed stream. SBF is not suitable for our propose either since it is not optimized for finding icebergs and does not afford us a tight tail bound on the estimation error.

In the following, we first present the scheme for a special case in which the count for every item at each node is either 0 or 1 (Section 4.1). This scheme is relatively easy to analyze and understand. We then extend this approach to a more complicated general case (Section 4.2).

4.1 0/1 Case

This is the case in which the frequency count for every item at each node is either 0 or 1. In this case the data at each node can be viewed as a set of items, and thus, the iceberg problem can be viewed as the problem of finding all the items appearing in more than T nodes. This special case occurs in real applications, such as the spyware detection problem discussed in Section 1.

The counting sketch we use to solve the special case is simply Bloom filters [8] combined with sampling. The Bloom filter is an ideal synopsis data structure for this 0-1 case since it is known to provide near-perfect lossy encoding of sets (see [23]). In our scheme, each node i samples its items with probability p and encodes the sampled items into a Bloom filter, denoted as B_i . As mentioned before, each node also samples a list of items A_i uniformly with probability p' , where p' is much smaller than p . After collecting (A_i, B_i) from all the nodes $i, i = 1, 2, \dots, N$, for each $I \in \cup_{i=1}^N A_i$, the server queries each Bloom filter whether it contains I and then calculates the total number of occurrences of I by scaling the total number of positive answers by $1/p$ and discounting the number of false positives that these filters might have provided.

We impose a natural homogeneous resource constraint on our system: the average amount of communication cost from each node i to the server, amortized over the number of items at node i , is approximately a constant D . We will show that this approach achieves a much smaller worst-case MSE and has much tighter tail bounds than the pure sampling-based approach when the same communication constraint is imposed. In the following, we first present a quick overview of the basic facts of the Bloom filter relevant to our encoding algorithm (Section 4.1.1). Then we present our unbiased estimator and derive its MSE (Section 4.1.2). Finally, we obtain a method for minimizing the MSE (Section 4.1.3).

4.1.1 Bloom filter basics

A Bloom filter is an approximate representation of a set S , which given an arbitrary element x , allows for the membership query “ $x \in S$?”. A Bloom filter employs an array B of m bits, initialized to all 0’s, and k independent hash functions h_1, h_2, \dots, h_k with range $\{1, \dots, m\}$. During *insertion*, given an element x to be inserted into a set S , the bits $B[h_i(x)], i = 1, 2, \dots, k$, are set to 1. To *query* for an element y , i.e., to check if y is in S , we check

the values of the bits $B[h_i(y)], i = 1, 2, \dots, k$. The answer to the query is *yes* if all these bits are 1, and *no* otherwise.

A Bloom filter guarantees not to have any false negative, i.e., returning “no” even though the set actually contains the item. However, it may contain false positives, i.e., returning “yes” while the item is not in the set. Let t be the number of items stored in the filter. Then the probability that the filter returns a false positive to the query about an item that is not stored is approximately

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kt}\right)^k \approx \left(1 - e^{-\frac{kt}{m}}\right)^k$$

(see [19]). The RHS is minimized to 2^{-k} when⁸ k is set to $m \ln 2/t$. We refer to the amount of storage consumed by each element, namely m/t , denoted as C . Then the optimal k value is equal to $C \ln 2$.

Our Bloom filter configuration for each node is homogeneous in the sense that they will sample their items with the same probability p , consume the same amount of storage per sampled item (translating into communication cost per item) C , and use the same (optimal) number of hash functions $k = C \ln 2$. This is consistent with our intention to impose the same constraint of communication cost per item D on each node. It is not hard to see that $D = C \times p$. Therefore, given D , there is a clear tradeoff between C , bits per sampled item, and p , the sampling rate. We will show in Section 4.1.3 how to configure C and p so that our estimator, to be discussed next, has the minimum MSE.

4.1.2 The estimator

After receiving the Bloom filter B_i and the sampled identities A_i from each node i , the server estimates the global count f_I of an item $I \in \cup_{i=1}^N A_i$ as follows: let α be the total number of Bloom filters that assert to contain I and let p be the sampling rate for generating (B_i) ’s. Let $q = 2^{-k}$ be the false positive rate of an individual Bloom filter. Since the sampling rate is p , out of the f_I occurrences pf_I are expected to be in the Bloom filters. All such Bloom filters should return a positive answer. The remaining $N - pf_I$ Bloom filters may return a false positive with false positive rate q . So, the expect number of positive answers is $pf_I + (N - pf_I)q$. Since α positive answers are returned, by solving $\alpha = pf_I + (N - pf_I)q$ for f_I , we obtain an estimator \hat{f}_I of f_I as

$$\hat{f}_I = \frac{\alpha - Nq}{p(1 - q)} \quad (3)$$

The following theorem shows that \hat{f}_I is unbiased, i.e., $E[\hat{f}_I] = f_I$. Its proof is omitted here due to limited space and can be found in [29]

THEOREM 3. (i) *The estimator \hat{f}_I is unbiased.* (ii) $MSE[f_I] = \text{Var}[\hat{f}_I] = \frac{f_I(1-2q-p+pq)}{p(1-q)} + \frac{qN}{p^2(1-q)}$.

4.1.3 Configuring for optimal performance and tail bound analysis

Recall when the communication cost per item is set to D , we need to configure parameters p and C , which satisfies $p \times C = D$ so that the MSE of our estimator is minimized. The interplay between p and C can be illustrated as follows. Larger C means that we devote more bits to each sampled item use a larger number of hash functions for encoding an item (recall that $k = C \ln 2$), which will reduce the false positive rate q (recall that $q = 2^{-k}$). However, this

⁸In this case, the Bloom filter contains approximately half 1’s and half 0’s.

implies smaller sampling rate p , which will increase the estimation error when sampling is compensated through scaling. Intuitively there is a sweet spot between two extremes. In this section, we develop the theory for finding this optimal spot.

Let $F(p)$ denote the MSE of our estimator \hat{f}_I as a function of p . The following theorem, the proof of which is quite involved and can be found in [29], shows how to minimize $F(p)$.

THEOREM 4. (i) *The first derivative of $F(p)$ has at most two roots in $[0, 1]$. (ii) *The optimal p is attained either at the smaller (or unique) root, or at $p = 1$.**

Like in the 0/1 case, a tight tail bound can be obtained by using Theorem 2. The parameter mapping is as follows. Again let S be the set of nodes at which I appears. For each $i \in S$, we set W_i in Theorem 2 to $\frac{Y_i - q}{p - pq} - 1$. Here Y_i takes value 1 if the Bloom filter B_i gives the positive answer regarding I , and takes value 0 otherwise. For each $i \notin S$, we set W_i to $\frac{Z_i - q}{p - pq}$. Here Z_i takes value 1 if B_i gives the positive answer regarding I , and takes value 0 otherwise. Thus, $|\frac{Y_i - q}{p - pq} - 1| \leq \frac{1 - q}{p - pq} - 1 = \frac{1}{p} - 1$ and $\frac{Z_i - q}{p - pq} \leq \frac{1 - q}{p - pq} = \frac{1}{p}$. Since $\frac{1}{p} > \frac{1}{p} - 1$ we set θ to $\frac{1}{p}$ in Theorem 2.

4.1.4 Numerical Results

Here we use an example to illustrate how much better this counting-sketch-based works than the pure sampling scheme. In this example, the length of an item identifier is assumed to be 100 bits (e.g., network flow label or URL), and there are $N = 10,000$ different nodes (same as the example in Section 3.3). We assume that the amount of bits sent to the server by any node can only be 0.02 of the raw data size at the node (i.e., 2 bits per item after compression on the average or $D = 2$ bits). Suppose there is one item I that occurs at exactly 5,000 nodes.

In the pure sampling scheme, each node samples 2.0% of the item (i.e., $d = 49$). In the sketch-based scheme, the sampling part samples 0.2% of the item identities, which is equivalent to the cost of 0.2 bits per item (each identity is 100 bits long). The Bloom filter part will cost 1.8 bits per item (recall that the total has to be 2). Compared to the sampling-based scheme the estimator here has much tighter tail bounds and smaller variance. For example, we have $\Pr[4887 \leq \hat{f}_I \leq 5,113] \geq 76\%$. In other words, with probability 0.76 the relative error is no more than 2.26%. In comparison, the sampling-based scheme can only guarantee, with probability 0.76 the relative error is no more than 20%. In addition, the standard deviation of the estimator here is only 60.3, about an order of magnitude smaller than in the sampling-based scheme (about 495).

4.2 The General Case

In the general case, the frequency count of an item could be any positive integer. Our approach is to extend the sketch-based method in Section 4.1 to obtain a solution to the general case. A naive solution here will be to divide the items into those having the same count and conquer each group using the scheme for the 0/1 case (the Bloom filter). It is not hard to see that this solution can be configured to generate an unbiased estimate as before. Unfortunately, preparing a Bloom filter for every possible count may not be very economical since more Bloom filters may result in more false positives and hence higher MSE.

We propose to resolve this by rounding each count to a point in a short series of quantization points $\{b_j\}_{j=1}^n$, $b_1 < \dots < b_n$. Then, at each quantization point, we build a Bloom filter to encode these (rounded) counts. Although this rounding (quantization) introduces an error, it helps to reduce the error caused by the false positives of the Bloom filters. The MSE of our estimator is in fact

the summation of these two types of errors. In this section, we analyze the interplay between these two types of errors to arrive at a near-optimal tradeoff.

In the 0/1 case, there is only one possible split pattern: split an aggregate count of x into x 1's. In the general case, however, there are many possible split patterns, since the count at each node may be greater than 1. Just like in our sampling solution, we need to consider the existence of an adversary that attempts to maximize the MSE by splitting the overall count. Our solution should be able to minimize the maximum (worst-case) MSE by carefully choosing the quantization points. Again, we resolve this by making all the split patterns equally bad for the adversary, in terms of having the same small overall MSE. The solution, however, is much more sophisticated than in the sampling case.

In the following we will provide the details of the scheme. We begin with a description of the encoding algorithm (Section 4.2.1). We then describe the decoding algorithm at the server, analyze its accuracy rigorously (Section 4.2.2), and show how to configure the system to minimize the worst-case MSE bound (Section 4.2.3). Finally we describe how the sampling module works to obtain the candidate item identities (Section 4.2.4).

4.2.1 The encoding algorithm

Suppose we are given a series of $n + 1$ “quantization points” $\{b_j\}_{j=0}^n$, $0 = b_0 < b_1 < b_2 < \dots < b_n = M$, chosen from the range of the global frequency counts⁹. These points are in fact algorithmically determined for the purpose of minimizing the worst-case MSE, but we will postpone the discussion of the algorithm (and the definition of M) until Section 4.2.3. A count of value v between 1 and b_n that is not equal to any quantization point, is probabilistically rounded to one of the quantization points as follows. Suppose $b_{j-1} \leq v \leq b_j$, $j \geq 1$. Then v is rounded to b_{j-1} with probability $\frac{b_j - v}{b_j - b_{j-1}}$ and to b_j with probability $\frac{v - b_{j-1}}{b_j - b_{j-1}}$. For example, if count 4 sits between two neighboring quantization points 3 and 7, then it is assigned to 3 with probability $\frac{3}{4}$ and to 7 with probability $\frac{1}{4}$. The rationale for adopting this quantization method is that an unbiased estimation of the original count can still be made after the quantization. If the count c of an identity I is larger than b_n , the largest value of the quantization points, the pair $\langle I, c \rangle$ will be sent to the server as it is (without any lossy encoding).

At each node i , for each j , $1 \leq j \leq n$, a Bloom filter $B_i^{(j)}$ is constructed to encode the items that are rounded to the quantization point b_j . Like in the 0/1 case, items rounded to b_j may first be subject to sampling and only the sampled items are encoded into the Bloom filter $B_i^{(j)}$. The question now is “What should be the resource consumption (in bits) of each sampled item and what should be the sampling rate in constructing $B_i^{(j)}$?”. To answer this question, we need to first decide “What should be the resource consumption of each item on the average, denoted as D_{b_j} , in constructing $B_i^{(j)}$?”. We refer to the mathematical rationale behind this decision (for $B_i^{(j)}$, $j = 1, 2, \dots, n$) as our resource consumption model. We need to establish this model for the general case, because unlike in the 0/1 case, item counts after quantization could take n possible positive values $b_1 < b_2 < \dots < b_n$, and clearly an item of larger count deserves more generous resource consumption. In other words, the larger the b_j value is, the larger the D_{b_j} value should be. But how should D_{b_j} grow as a function of b_j ?

Interestingly, the answer to this second question is intertwined (but without “chicken and egg problem”) with the answer to the first question. The intuition of our resource consumption model

⁹The point b_0 is a virtual point to simplify our description.

is again to make each split pattern result in the same overall MSE (the independence principle) like in the sampling-based scheme. In other words, we set D_{b_j} to such a value that the MSE in estimating a fragment of size b_j from the Bloom filter $B_i^{(j)}$ is exactly $d * b_j$ (i.e., proportional to b_j), where d is a constant. This MSE in turn is computed from the optimal tradeoff between the resource consumption of a sampled item, denoted as C_j , and the sampling probability, denoted as p_j , under the constraint $C_j * p_j = D_{b_j}$. As a consequence, the optimal sampling rate p_j may be different at different (possible) quantization values b_j 's. The technique in determining this optimal tradeoff is similar to that we described in Section 4.1.3. Since we do not know the values of b_j at this point yet, when we compute this resource consumption model, we assume trivial quantization, i.e., every count value is a quantization point. In other words, we compute D_c for every possible quantization value c based on the above independence principle and obtain the aforementioned optimal tradeoff point on this quantization value c as a byproduct of this computation. These computation results will serve as constant parameters (hence no ‘‘chicken and egg’’ problem) to our algorithm in determining the optimal quantization points b_1, b_2, \dots, b_n , shown later in Algorithm 1.

4.2.2 The decoding algorithm

The central server will collect from each node i the n Bloom filters $\{B_i^{(j)}\}_{j=1}^n$ and the set A_i of sampled item identities (produced by a separate sampling module to be described later). As in the 0/1 case, each item in $\cup_{i=1}^N A_i$ is used to query the Bloom filters and the results are used to estimate the global count of the item. Let I be an item belonging to $\cup_{i=1}^N A_i$. For each j , $1 \leq j \leq n$, let g_j be the random variable that takes as value the number of nodes at which the count for I (its global count denoted as f_I) is rounded to the quantization point b_j . We are able to obtain an unbiased estimator \hat{g}_j of g_j derived from Equation (3) in the 0/1 case, when we consider the Bloom filters at the quantization point b_j across all the nodes. These (g_j) 's, viewed as a random vector, is in fact a (random) function of the split pattern of f_I . We obtain an estimator of f_I as a functions of (g_j) 's:

$$\hat{f}_I = \sum_{j=1}^n b_j \hat{g}_j. \quad (4)$$

This is an unbiased estimator. The proof can be found in [29].

THEOREM 5. $\hat{f}_I = \sum_{j=1}^n b_j \hat{g}_j$ is unbiased.

4.2.3 Configuring parameters for minimizing worst-case MSE

The following theorem establishes a tight bound of the MSE of our estimator as a function of the quantization points and the split pattern of f_I . Its proof can also be found in [29].

THEOREM 6. $\text{Var}[f_I] \leq \sum_{j=1}^n F_j(e_j)$, where for each j , $1 \leq j \leq n$, $F_j(e_j)$ is a linear function of the form $a_j e_j + d_j$, such that

$$\begin{aligned} e_j &= b_j \mathbb{E}[g_j] \\ a_j &= \frac{b_j(1 - 2q_j - p_j + p_j q_j)}{p_j(1 - q_j)} \\ &\quad + \frac{\max\{(b_j - b_{j-1})^2, (b_{j+1} - b_j)^2\}}{4b_j} \\ d_j &= \frac{q_j N b_j^2}{p_j^2(1 - q_j)} \end{aligned}$$

where b_{n+1} is set to b_n by convention (for computing a_n).

Now based on Theorem 6, we tune the quantization points to achieve the near-optimal accuracy. Note that e_1, e_2, \dots, e_n can be viewed as a ‘‘quantized split pattern’’, since they correspond to the average number of fragments that are quantized to b_1, b_2, \dots, b_n by rounding, respectively, and $\sum_{j=1}^n e_j = f_I$. The variance of our estimator, as we show in Theorem 6, is bounded by $\sum_{j=1}^n (a_j e_j + d_j)$. Again like in Section 3.2, we imagine that an adversary will try to split f_I into (e_i) 's such a way that maximizes this MSE. Note that for any such split, the values (a_j) 's and (d_j) 's remain constant. Suppose the (a_j) 's are ordered as $a_{\pi(1)} \geq a_{\pi(2)} \geq \dots \geq a_{\pi(n)}$, where π is a permutation defined over $\{1, 2, \dots, n\}$. Then the best strategy for the adversary is to manipulate the split pattern so that $e_{\pi(1)}$ is first made as large as possible, and then $e_{\pi(2)}, e_{\pi(3)}$ and so on, until the total f_I has been reached (i.e., the greedy strategy). We would like to configure our parameters to make this worst MSE as small as possible.

Our strategy, like in the sampling case, is to first make the constant factor a_i on each e_i piece approximately equal to a . Then no matter how the adversary splits f_I , the MSE of our estimation is approximately equal to $a f_I + \sum_{j=1}^n d_j$, a linear function of f_I . There is a fundamental tradeoff between a , the slope of the function, and $\sum_{j=1}^n d_j$, the intercept of the function. This tradeoff is in fact the result of the interplay between the quantization error, and the errors caused by Bloom filter false positives and sampling. When the distance between neighboring quantization point pairs decreases, the slope of the function, which corresponds to the quantization error, also decreases, but the intercept, which corresponds to the errors caused by Bloom filter false positives and sampling, increases since more quantization points result in more false positives. In the following, we develop an algorithm to further tune the quantization points to find the best tradeoff point between a and (d_j) 's so that $a f_I + \sum_{j=1}^n d_j$ is minimized (while keeping (a_i) 's close to each other).

Algorithm 1: Algorithm for Computing Quantization Points.

```

1 Input:  $N, M, T$ 
2 Output:  $b_j, j = 1, 2, \dots$ 
3 Optimize:
4  $b_0 \leftarrow 0; b_1 \leftarrow 1$ 
5 for each  $k, 2 \leq k \leq M$ 
6  $b_2 \leftarrow k$ 
7 Compute  $a_1$  and  $d_1$ 
8 /*  $a_j$  and  $d_j, j = 1, 2, \dots$ , are computed by Theorem 6 */
9  $j \leftarrow 3$ 
10  $l \leftarrow b_2$ 
11 while  $l < M$ 
12  $b_j \leftarrow \text{argmin}_{l < b_j \leq M} |a_1 - a_{j-1}|$ 
13  $l \leftarrow b_j$ 
14  $j \leftarrow j + 1$ 
15  $v_i \leftarrow \max\{a_1, a_2, \dots\} \times T + \sum_j d_j$ 
16 Find  $i$  whose  $v_i$  is the smallest of all
17 Output the  $(b_j)$ 's with respect to  $i$  as the quantization points

```

The pseudo-code of the algorithm is shown above (Algorithm 1). The intuition behind this algorithm is as follows. It takes as input three parameters, namely, N , the number of nodes, T , the iceberg threshold, and M , the cutoff point such that all items with counts larger than the point will be reported to the server (together with their counts) in the raw form (without Bloom filter encoding). Recall that our goal is to make the slope on each quantization point approximately same. The value a_1 will be determined by the choice of b_0 (fixed to 0), b_1 (fixed to 1), and b_2 . So, as soon as b_2 is fixed to some value ≥ 2 , the value of a_1 is set. The next quantization point b_3 is chosen so that a_2 is close to a_1 . We repeat

Trace	# of nodes	# of item-count pairs
NetFlow traces	216	2,927,878
IBM system logs	32	1,982

Table 1: Data sets used in our experiments.

this to find b_4, b_5, \dots until we reach the cutoff point M . These particular choices were obtained by fixing b_2 . So, by changing the value of b_2 a difference series of quantization points can be obtained. We try each possible value (from 2 to M) for b_2 and compute the series. We choose the value for b_2 that minimizes the linear function $af_I + \sum_{j=1}^n d_j$ (computed according to Theorem 6). The complexity of the above algorithm is $O(M^2 \log M)$ (there are two nested loops and the computation of argmin uses binary search algorithm.). Notice that this algorithm needs to be executed only once during the system configuration. Since M is usually no more than a few hundred in most of the targeted applications, the actual running time of this algorithm is very short. We will show two examples of spectrum of the quantization points in Section 5 generated by this algorithm using real-world data sets.

4.2.4 Sampling module

Recall that we must define a method for sampling item identities to generate lists A_1, \dots, A_N . As in our sampling-based scheme we use a size-dependent sampling technique so as to give a high overall sampling rate to an item with a higher total count. To do so, we propose to sample an item with count v at a node with probability $1 - e^{-\gamma v}$ for a constant γ . If the counts of an item I in the nodes are x_1, \dots, x_N , then the probability that it is not sampled at any node is $\prod_{i=1}^N e^{-\gamma x_i} = e^{-\gamma f_I}$. So, the probability it is sampled at at least one node is $1 - e^{-\gamma f_I}$. We can tune the parameter γ so as to achieve a certain success probability. For example, suppose there is a distributed iceberg whose global count is 10,000. If we set γ to 0.002, the probability its identity is not sampled at any node is only $e^{-20} = 2.06 \times 10^{-9}$. Finally, note that in the 0/1 case the value of v is either 0 or 1 this sampling scheme becomes uniform sampling.

5. EVALUATION

In this section we evaluate our proposed counting-sketch-based scheme using real-world data sets. We do not evaluate the sampling approach since it has been shown to be less accurate through analysis earlier.

5.1 Data Sets

Our motivating applications include detecting DDoS attacks and monitoring “hot spots” in large scale distributed systems. For the first type of the application, we apply our scheme on Internet2 traffic logs [3] to identify hosts (destination IP addresses) that receive large number of flows. For the second type, we study the system event logs of an IBM network to identify frequent system events generated by a number of different hosts.

The Internet2 traffic traces [3] were anonymized NetFlow [2] data collected from nine core routers in the Abilene network. The data represents one full day of router operation, broken into 288 five-minute epochs. we divided the data from each router in a random fashion to simulate an environment of 216 different nodes. The system event logs were collected from 32 hosts in a production network at IBM. Each machine logs the name of the event and the timestamp of its occurrence. Table 1 summarizes all the data sets used in the evaluation. All of our experiments were carried out on a 2.8 GHz Dell PowerEdge workstation running Linux Kernel version 2.2.21.

Trace	original (KB)	our scheme (KB)
NetFlow traces	22,203.4	1576.4
IBM system logs	35.8	2.05

Table 2: Communication cost of our scheme.

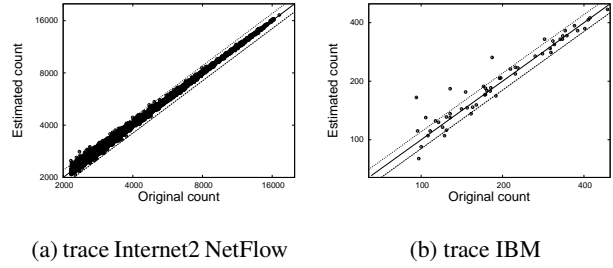


Figure 1: Original (x-axis) vs. estimated (y-axis) global frequency counts by the counting-sketch-based scheme. Notice both axes are on logscale.

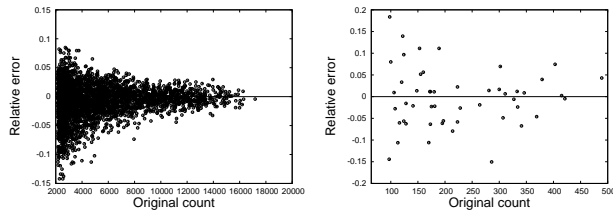
5.2 Experiment Setup and Results

For both experiments we set the largest quantization point to 100 and remove all occurrences of items whose local frequency counts are over 100 at each node, since such $\langle \text{item}, \text{count} \rangle$ pairs will be delivered to the server directly (instead of being hashed into the Bloom filters) with zero information loss. Therefore, our experiments reflect precisely the accuracy of our counting sketch estimator.

For the experiments on the Internet2 NetFlow traces, we set the communication cost to 2 bits for each item-count pair whose count is 1. The iceberg threshold T is set at 2,160, which corresponds to 10 occurrences per node on the average. According to the aforementioned resource constraint model, we compute the communication “budget” for items with higher count values. The cost per pair gradually increases from 3 bits to 15.1 bits. Then using Algorithm 1 we compute the spectrum of quantization points as $\{1, 4, 10, 19, 32, 48, 67, 89, 100\}$. The slopes $((a_j)$ ’s) generated by the algorithm are very close to each other (all around 1.63). Small discrepancies among slopes are unavoidable due to our constraint that quantization points have to fall on integers. Similarly, for the experiments on the IBM system logs, we set the iceberg threshold T to 96, which corresponds to 3 occurrences per node on the average. The communication cost per item-count pair gradually increases from 5 bits to 21.6 bits. Then the spectrum of quantization points is computed as $\{1, 3, 6, 11, 17, 25, 35, 46, 59, 74, 90, 100\}$ and the slopes are all around 0.90.

In Internet2 traces because each item is identified by a destination IP address, which is 32 bits, and the count for each item also needs 32 bits, each item-count pair occupies 64 bits. And in IBM system logs each system event is denoted by a long character string (around 16 characters on average in the logs we have) and each count also has 32 bits. Thus each item-count pair in the logs is 160 bits on the average. Table 2 compares the size of the original data bags (not original data streams) with the communication cost our scheme uses. It is observed that our scheme achieves very efficient communication bandwidth usage compared with the naive approach of shipping all the data bags to the central server. The data reduction ratio of the Internet2 NetFlow traces is about 14 to 1 and the ratio is about 17 to 1 for IBM system logs. We will show next that even with such a small communication cost our scheme allows us to achieve very accurate estimations.

Figures 1(a) and 1(b) compare the global frequency counts es-



(a) trace Internet2 NetFlow

(b) trace IBM

Figure 2: Relative errors of our estimations.

estimated using our proposed sketch-based scheme with their actual values. In both figures, the solid diagonal line denotes perfect estimation, while the dashed lines that parallels the solid line denote estimations of relative error $\pm 10\%$. Clearly, the closer the points cluster around the diagonal line, the more accurate the estimations are. We only plot the items whose frequency counts are estimated to be larger than or equal to the pre-defined thresholds. We observe that estimations are very accurate in both experiments, and that the estimations are more accurate for relatively large frequency count values (in Figure 1(a) the points cluster closer around the diagonal line when their values become large.)

Figures 2(a) and 2(b) quantify the accuracy of our estimations shown in Figure 1 respectively, using the relative error $(\frac{\hat{f}_i - f_i}{f_i})$ as the evaluation metric. In both figures, points above and below the solid line in the middle (perfect estimation) denote overestimates and underestimates respectively. In Figure 2(a), the points are scattered around the solid line in an approximately symmetric way, demonstrating the unbiased nature of our estimator. The same symmetry also can be found in Figure 2(b) but is not that clear since there are not many points there. The relative errors are mostly within ± 0.05 in Figure 2(a), and within ± 0.1 in Figure 2(b). In both experiments, the relative error generally decreases when the actual count value increases. This is more obvious in Figure 2(a) since it has more points.

6. CONCLUSION

Identifying icebergs in distributed datasets with minimum communication overhead is an open problem. The problem has applications in many areas ranging from network monitoring to bio-surveillance. To the best of our knowledge, this paper presents the first work on finding global icebergs in a distributed environment without assuming that a globally frequent item is also locally frequent. We propose two solutions to the global iceberg problem, one based on sampling, and the other based on a novel counting sketch. We show that our sampling strategy is near-optimal, resulting in almost the lowest possible estimation error under certain resource (communication cost) constraints. Our second approach uses a novel counting sketch to help detect icebergs in a much more communication-efficient way than the sampling-based solution. Theoretical and experimental analysis demonstrate the statistical properties of our proposed algorithms and their high accuracy.

7. REFERENCES

- [1] Akamai technologies inc. <http://www.akamai.com/>.
- [2] Cisco netflow. <http://www.cisco.com/warp/public/732/netflow/>.
- [3] Internet2 abilene network. <http://abilene.internet2.edu/>.
- [4] N. Alon and J. H. Spencer. *The Probabilistic Method*. A Wiley-Interscience Publication, 2000.
- [5] A. Arasu and G. S. Manku. Approximate quantiles and frequency counts over sliding windows. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2004.
- [6] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain : Operator scheduling for memory minimization in data stream systems. In *Proc. ACM SIGMOD*, June 2003.
- [7] B. Babcock and C. Olston. Distributed Top-K monitoring. In *Proceedings of ACM SIGMOD international conference on Management of data*, 2003.
- [8] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the Association for Computing Machinery*, 13(7):422–426, 1970.
- [9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. 29th ICALP*, July 2002.
- [10] C. Chen and Y. Ling. A sampling-based estimator for top-k query. In *Proc. International Conference on Data Engineering (ICDE)*, 2002.
- [11] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Proc. CIDR - Biennial Conference on Innovative Data Systems Research*, January 2003.
- [12] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Stat.*, 23:493–509.
- [13] S. Cohen and Y. Matias. Spectral bloom filters. In *Proc. ACM SIGMOD Conference on Management of Data*, 2003.
- [14] S. Cohen and Y. Matias. Spectral bloom filters. In *Proc. ACM SIGMOD Conference on Management of Data*, 2003.
- [15] G. Cormode and S. Muthukrishnan. What’s hot and what’s not: Tracking most frequent items dynamically. In *Proc. ACM PODS*, 2003.
- [16] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, April 2005.
- [17] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *Proc. VLDB*, 2004.
- [18] E. Demaine, J. Munro, and A. Lopez-Ortiz. Frequency estimation of internet packet streams with limited space. In *Proc. European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany*, 2002.
- [19] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.
- [20] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. In *Proc. the 24th International Conference on Very Large Data Bases (VLDB)*, 1998.
- [21] J. Feigenbaum and S. Kannan. Streaming algorithms for distributed, massive data sets. In *In proceedings of FOCS*, 1999.
- [22] R. Karp, S. Shenker, and C. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28:51–55, 2003.
- [23] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proc. ACM SIGMETRICS*, June 2004.
- [24] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-Code Bloom Filter for Efficient per-flow Traffic Measurement. In *Proc. IEEE INFOCOM*, March 2004. Extended abstract appeared in *Proc. ACM IMC* 2003.
- [25] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proc. International Conference on Data Engineering (ICDE)*, 2005.
- [26] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, 2002.
- [27] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. ACM SIGMOD*, June 2003.
- [28] Y. Wang. Personal communication. Dec 2004.
- [29] Q. Zhao, M. Ogihara, H. Wang, and J. Xu. Finding global icebergs over distributed data sets. In *Technical report*, April 2006.