Estimating the Frequency of Data Items in Massive Distributed Streams

Emmanuelle Anceaume, Yann Busnel, Nicolò Rivetti

Estimating the Frequency of Data Items in Massive Distributed Streams

Emmanuelle Anceaume
IRISA / CNRS
Rennes, France
Emmanuelle.Anceaume@irisa.fr

Yann Busnel
Crest (Ensai) / Inria
Rennes, France
Yann.Busnel@ensai.fr

Nicolò Rivetti
LINA / Université de Nantes
Nantes, France
Nicolo.Rivetti@univ-nantes.fr

***Abstract*—We investigate the problem of estimating on the fly the frequency at which items recur in large scale distributed data streams, which has become the norm in cloud-based application. This paper presents CASE, a combination of tools and probabilistic algorithms from the data streaming model. In this model, functions are estimated on a huge sequence of data items, in an online fashion, and with a very small amount of memory with respect to both the size of the input stream and the values domain from which data items are drawn. We derive upper and lower bounds on the quality of CASE, improving upon the Count-Min sketch algorithm which has, so far, been the best algorithm in terms of space and time performance to estimate the frequency of data items. We prove that CASE guarantees an (ε, δ)-approximation of the frequency of all the items, provided they are not rare, in a space-efficient way and for any input stream. Experiments on both synthetic and real datasets confirm our analysis.**

***Index Terms*—Data stream model; frequency estimation; randomized approximation algorithm.**

I. Introduction

Huge data flows have become very common in the last decade. This has motivated the design of online algorithms that allow the accurate estimation of statistics on very large data flows. A rich body of algorithms and techniques have been proposed for the past several years to efficiently compute statistics on massive data streams. In particular, estimating the number of times data items recur in data streams in real time enables, for example, the detection of worms and denial of service attacks in intrusion detection services, or the traffic monitoring in cloud computing applications.

Two main approaches exist to monitor in real time massive data streams. The first one consists in regularly sampling the input streams so that only a limited amount of data items is locally kept. This allows to exactly compute functions on these samples. However, accuracy of this computation with respect to the stream in its entirety fully depends on the volume of data items that has been sampled and their order in the stream. Furthermore, an adversary may easily take advantage of the sampling policy to hide its attacks among data items that are not sampled, or in a way that prevents its "malicious" data items from being correlated [1]. In contrast, the streaming approach consists in scanning each piece of data of the input stream on the fly, and in locally keeping only compact synopses or *sketches* that contain the most important information about these data. This approach enables us to derive some data streams statistics with guaranteed error bounds without making any assumptions on the order in which data items are received at nodes. Sketches highly rely on the properties of hashing functions to extract statistics from them. Sketches vary according to the number of hash functions they use, and the type of operations they use to extract statistics.

In this paper we focus on the *Count-Min sketch* algorithm proposed by Cormode and Muthukrishnan [2]. This data structure so far predominates all the other ones in terms of space and time needed to guarantee an additive ε-accuracy on the estimation of item frequencies. Briefly, this technique performs t random projections of the set of items of the input stream into a much smaller co-domain of size k, with $k = \lceil e/\varepsilon \rceil$ and $t = \lceil \log(1/\delta) \rceil$ where $0 < \varepsilon, \delta < 1$. The user defined parameters ε and δ represent respectively the accuracy of the approximation, and the probability with which the accuracy holds. However, because k is typically much smaller than the total number of distinct items in the input stream, hash collisions do occur. This affects the estimation of item frequency when the size of the stream is large. More precisely, this significantly biases the frequency estimation of all the items except those which are very frequent with respect to the other ones. To mitigate this saturation issue two main approaches have been proposed. Dimitropoulos *et al.* [3] keep only fresh items of the input stream so that the number of distinct items that appear in the sketch does not increase with the size of the stream. While effective to decrease the number of collisions, such an approach opens the door to adversarial behaviors, and in particular dormant attacks, where the adversary from time to time floods the stream with its own items, eclipsing accordingly most of the other items in the sketch. The second approach, proposed by Estan and Varghese [4], consists in limiting the number of updates by first updating the smallest of the counters normally, and by setting the other counters to the maximal of their old value and the new value of the smallest counter. Such an approach requires that only items with positive values are received.

In this paper, we present an alternative approach to reduce the impact of collisions on the estimation of item frequency. The intuition of our idea is that by keeping track of the most frequent items of the stream, and by removing their weight from the one of the items with which these frequent items collide, the over-estimation of non frequent items is drastically decreased. We show that our algorithm, called CASE for Count-Any Sketch Estimator for Frequency Approximation, significantly improves upon the CM sketch by proving that our solution provides an (ε, δ)-multiplicative approximation of the frequency of items that recur sufficiently often in the stream. No matter how items are distributed

This work was partially funded by the French ANR project SocioPlug (ANR-13-INFR-0003), and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

in the streams, the quality of our solution remains unchanged. In particular, we guarantee that for any ε, δ in $(0,1)$, our algorithm (ε,δ)-approximates item frequency by using small workspace, *i.e.*, $O\left((b+(\log(1/\delta)/\varepsilon)(b+1/\varepsilon))\left(\log m+\log n\right)\right)$ bits of space, where m represents the total number of items read so far in the stream, n is the number of distinct data items among the m ones and, b represents the number of the most frequent items monitored. For very rare items, an additive approximation of their frequency is only provided. We then propose to judiciously distribute this local sketch to estimate the global frequency of any item that may recur in multiple streams. This distributed algorithm organizes nodes of the system in a distributed hash table (DHT) such that each node implements a tiny local sketch on a reduced number of items. By doing so we guarantee a greater accuracy of the estimation of item frequency. Finally, simulations both on synthetic and real traces that exhibit skews in various ways confirm the accuracy of CASE. To the best of our knowledge, we are not aware of any comparable work.

The remaining of the paper is organized as follows. Section II presents the model this work relies on. Section III describes and analyses CASE, the sketch we propose to mitigate the impact of collisions on the estimation of item frequency. Section IV presents a simple way to distribute sketches among all the nodes so that distributed streams can be efficiently monitored. Section V evaluates the minimum effort the adversary needs to exert to bias item frequency estimation. Finally, the main results of extended simulations are presented in Section VI. Section VII concludes.

II. Model and Background

A. Model

We present the computation model under which we analyze our algorithms. We consider a large scale system $\mathcal{S}$, such that each node $i \in \mathcal{S}$ receives a large sequence $\sigma^{(i)}$ of data items or symbols $< u, v, w, \ldots >$ drawn from a very large universe N (these data items can for example model TCP/IP packets, or HTTP requests [5]). In the following we denote the different items by integers $1, \ldots, n$, with $n = |N|$. The number of times item u appears in a stream is called the frequency of item u, and is denoted by f_u. Items arrive regularly and quickly, and due to memory constraints, items must be processed sequentially and in an online manner. In the following we only focus on the frequency estimations of the items that effectively appear in the stream [6]. We refer the reader to [7] for a detailed description of data streaming models and algorithms.

B. Adversary

We assume the presence of malicious (*i.e.*, Byzantine) nodes that collectively try to subvert the system by manipulating the prescribed algorithms. We model these behaviors through an adversary that fully controls and manipulates these malicious nodes. We suppose that the adversary is strong in the sense that it may actively tamper with the data stream of any node $i \in \mathcal{S}$ by observing, and injecting a potentially large number ℓ of items. Indeed, the goal of the adversary is to judiciously increase the frequency of its ℓ items to bias the frequency estimation of the items generated by correct nodes. The number ℓ is chosen by the adversary according to the parameters of the sketches at correct nodes. By *correct*, we mean a node present in the system which is not malicious. Note that correct nodes cannot *a priori* distinguish items sent by correct nodes from the ones sent by malicious ones. Classically, we assume that the adversary can neither drop a message exchanged between two correct nodes nor tamper with its content without being detected. This is achieved by assuming the existence of a signature scheme (and the corresponding public-key infrastructure) ensuring the authenticity and integrity of messages. This refers to the authenticated Byzantine failure model [8]. We finally suppose that any algorithm run by any correct node is public knowledge to avoid some kind of security by obscurity. However the adversary has not access to the local random coins used in the algorithms.

C. Preliminaries

We present background on data streams analysis that make this paper self-contained.

a) 2-universal Hash Functions: In the following, we use hash functions randomly picked from a 2-universal hash functions family. A collection $\mathcal{H}$ of hash functions $h : \{1, \ldots, M\} \to \{0, \ldots, M'\}$ is said to be *2-universal* if for every two different items $x, y \in [M]$, $\mathbb{P}_{h\in\mathcal{H}}\{h(x) = h(y)\} \leq \frac{1}{M'}$, which is the probability of collision obtained if the hash function assigned truly random values to any $x \in [M]$.

b) Randomized (ε,δ)-approximation Algorithm: A randomized algorithm $\mathcal{A}$ is said to be an (ε,δ)-approximation of a function ϕ on σ if for any sequence of items in the input stream σ, $\mathcal{A}$ outputs $\hat{\phi}$ such that $\mathbb{P}\{\mid \hat{\phi} - \phi \mid > \varepsilon\phi\} < \delta$, where $\varepsilon, \delta > 0$ are given as parameters of the algorithm.

c) Frequency moments: Frequency moments are important statistical tools that have been introduced by Alon *et al.* [9]. Computing frequency moments F_j allows us to quantify the amount of skew in a data stream. For each $j \geq 0$, the j-th frequency moment F_j of σ is defined as $F_j = \sum_{v\in N} f_v^k$, where f_v represents the number of occurrences of v in the stream. Among the remarkable moments, F_0 represents the number of distinct items in a stream while F_1 corresponds to the size of the stream.

III. The CASE algorithm

In this section, we describe how we improve upon the Count-Min sketch [2] to get an (ε,δ)-multiplicative approximation of the frequency of items that recur sufficiently often in the stream. The intuition is that by keeping track of the most frequent items of the stream, and by removing their weight from the one of the items with which these frequent items collide in the sketch, the over-estimation of item frequency is limited. Prior to presenting the Count-Min sketch and our extension, we first describe two algorithms that our solution uses as building blocks. One is due to Bar-Yossef *et al.* [10] to estimate F_0, the number of distinct items in the stream, while the second one has been proposed by Metwally *et al.* [11] to determine the most frequent items in a stream. Note that both algorithms are data-stream algorithms.

A. Estimating F_0, the number of distinct items in the stream

The problem of estimating the number of distinct elements has received a lot of attention in the data stream model. First, the seminal work of Flajolet and Martin [12] has shown that it is possible to compute such an estimate using only logarithmic space in F_0 by relying on properties of hash functions. Afterwards, follow-up

Algorithm 1: BJKST algorithm

Input: An input stream σ; r_1 and r_2 settings;
Output: The estimate $\hat{F}_0$ of the number of distinct elements in the stream

```
1  Choose r_1 2-universal hash functions h : [n] → [n];
2  Choose r_1 2-universal hash functions g : [n] → [r_2];
3  Initialization of r_1 buffers B_j of size r_2;
4  for j ∈ [1 ... r_1] do ξ_j = 0; B_j = ∅
5  for a_i ∈ σ do
6      v = a_i;
7      for j ∈ [1 ... r_1] do
8          z = zeros(h_j(v));
9          if z ≥ ξ_j and g(v, z) ∉ B_j then
10             B_j = B_j ∪ {g(v, z)};
11             while | B_j |> s do
12                 B_j = B_j \ {g(v', z')} with z' = ξ_j;
13                 ξ_j = ξ_j + 1;
14 return F̂_0 = median_{1≤j≤r} 2^{ξ_j} |B_j|;
```

enhancements have improved the accuracy of the estimation [10][1]. Thereafter, we briefly sketch the BJKST algorithm proposed by Bar-Yossef *et al.* [10]. The BJKST algorithm is based on the coordinating sampling algorithm of Gibbons and Tirthapura [14]. The BJKST algorithm, whose pseudo-code is given in Algorithm 1, consists in running r instances of the same procedure, such that procedure $j \in \{1, \ldots, r\}$ uses hash function h_j. The hash function h_j determines the "level" of items from the stream such that half of the items have a level equal to 1, a quarter of them have a level equal to 2, ..., until finally $\frac{1}{2^i}$ of them have a level equal to i. This level is provided by the function *zeros(v)* that returns the number of trailing zeros in the binary representation of v.

Theorem 1 ([10]) *The Bar-Yossef* et al. *[10] algorithm with parameters ε and δ outputs $\hat{F}_0$ such that $\mathbb{P}\{|\hat{F}_0 - F_0| > \varepsilon\} < \delta$. The worst-case running time for each input symbol is $\mathcal{O}(\log(1/\delta)/\varepsilon^2)$, and the total space required by the algorithm is $\mathcal{O}((\log(1/\delta)\log n)/\varepsilon^2)$ bits.*

B. Determining the most frequent data items

The problem of determining the most frequent items in a stream has also been extensively studied in the data stream literature, and in particular by Misra and Gries [15], and then by Metwally *et al.* [11]. Thereafter, we recall the main principles of the algorithm of Metwally *et al.* [11]. This algorithm, called Space Saving, takes two parameters: ε and b, such that $0 < 1/b \leq \varepsilon \leq 1$. It maintains b $\langle tuple, counter \rangle$ couples, and returns all items whose frequency of occurrences are greater than or equal to $m\varepsilon$. This algorithm is deterministic. Initially, all the b counters are set to $(\perp, 0)$. Upon receipt of an item v, if v is already monitored, the associated counter is incremented. In the negative and if there exists a free tuple, then it is allocated to v and its counter is set to the minimum value of all the counters plus 1. Finally, if there is no more free tuple, then v replaces the item associated to the counter with the minimum value (if there are several such items, then one of them is randomly chosen) and the associated counter is incremented by one.

[1]A comprehensive survey describing the literature on distinct elements in this model is presented by Gibbons in [13].

Theorem 2 ([11]) *The Space Saving algorithm with parameters b and ε, with $b \geq 1/\varepsilon > 1$, returns for any item v an estimate $\hat{f}_v^{(\mathtt{SS})}$ such that $f_v - m\varepsilon \leq \hat{f}_v^{(\mathtt{SS})} \leq f_v$ with $\mathcal{O}(b(\log m + \log n))$ bits of space.*

C. Estimating the frequency of each data item

Cormode and Muthukrishnan [2] propose a probabilistic data structure, the Count-Min sketch (CM), that allows the computation, for any item v read from the input stream σ, of an estimation $\hat{f}_v^{(\mathtt{CM})}$ of the number of times v has occurred since the inception of the stream. The estimation is computed by maintaining a two-dimensional array C of $t \times k$ counters and by using a collection of 2-universal hash functions $\{h_1, \ldots, h_t\}$. Each time an item v is read from the input stream, this causes one counter per line to be incremented. When a query is issued to get $\hat{f}_v^{(\mathtt{CM})}$ the returned value is equal to the minimum among the t values $C[i][h_i(v)]$ for $i = 1, \ldots, t$. This sketch has so far been considered as the best one in terms of space and time performance to estimate data item frequency.

Theorem 3 ([2]) *The Count-Min sketch algorithm with parameters $k = \lceil e/\varepsilon \rceil$ and $t = \lceil \log(1/\delta) \rceil$ returns for any item v in σ an estimate $\hat{f}_v^{(\mathtt{CM})}$ such that $\mathbb{P}\{|\ \hat{f}_v^{(\mathtt{CM})} - f_v\ | > \varepsilon(m - f_v)\} < \delta$ with $\mathcal{O}(\log(1/\delta)(\log m + \log n)/\varepsilon)$ bits of space.*

D. The Count-Any Sketch Estimator (CASE)

Our algorithm works by partitioning all the items uniformly at random into groups, by estimating their frequency, and then by removing the weight of heavy hitters (*a.k.a* very frequent items), computed with the Space Saving algorithm, from the group of items with which heavy hitters collide. Indeed, in the CM algorithm, because k is typically much smaller than the item identifiers space, hash collisions do occur. This significantly affects the accuracy of the estimation when items frequently recur in the input stream, that is when the size m of the input stream becomes large. Our approach guarantees that the error of the estimator in answering a query for $\hat{f}_v$ is within an error εf_v with probability $1 - \delta$ for non rare items. Algorithm 2 presents the pseudo-code of CASE. The update phase (Lines 5 to 13), made of Tasks T_1, T_2, and T_3 run in parallel, updates the three data-structures described previously upon receipt of items from the stream, while the query phase (Lines 14 to 17) returns the frequency estimation of any item v after having removed, if needed, the over-estimation introduced by highly frequent items that collide with v.

We prove below that for any ε and δ, and for any non rare data item v read from the input stream, CASE gives an (ε, δ)-approximation of f_v. The following theorem proceeds in three steps to derive the quality of the estimation. Namely, the estimation is derived for highly frequent items, for frequent items and for rare items. In the two former case, (ε, δ)-approximation is guaranteed, while in the latter one, an additive (ε, δ)-approximation is ensured.

Theorem 4 *CASE, run with parameters $k = \lceil 2(b-1)/\varepsilon \rceil$, $t = \lceil \log(1/\delta) \rceil$ and $1 < b \leq 4/\varepsilon^3$, returns for any item u an estimate $\hat{f}_u$ such that*

$$\mathbb{P}\left\{|\hat{f}_u - f_u| \geq \varepsilon f_u\right\} \leq \delta \quad \textit{if} \quad \forall v \in N, f_v \leq \frac{\varepsilon^2 m}{2k}$$

Algorithm 2: Count-Any Sketch Estimator

Input: An input stream σ, precision parameters ε, δ, b;
Output: Estimation of the frequency of any item v in σ

1 $k \leftarrow \lceil 2(b-1)/\varepsilon \rceil$;
2 $t \leftarrow \lceil \log(1/\delta) \rceil$;
3 $m = 0$;
4 Choose t 2-universal hash functions $h_1, \ldots, h_t : [|\Omega|] \to [k]$;
5 **for** $v \in \sigma$ **do**
6 $m \leftarrow m + 1$;
7 **Task** T_1 **:**
8 **for** $i = 1$ **to** t **do**
9 $C[i][h_i(v)] \leftarrow C[i][h_i(v)] + 1$;
10 **Task** T_2 **:**
11 Update B with v according to the Space Saving algorithm initialized with parameters ε and b;
12 **Task** T_3 **:**
13 Update F_0 with v according to the BJKST algorithm initialized with parameters $1/\varepsilon^2$ and $\log(1/\delta)$;
14 **Upon query of** $\hat{f}_u$ **:**
15 $\hat{f}_u^{(\mathtt{CM})} \leftarrow \min_{1 \le i \le t} (C[i][h_i(u)])$;
16 **if** $u \in B$ **and** $\hat{f}_u^{(\mathtt{CM})} \ge m\varepsilon$ **then return** $\hat{f}_u^{(\mathtt{CM})}$ **else return** $\hat{f}_u^{(\mathtt{CM})} \cdot k/F_0$

or

$$\begin{cases} \mathbb{P}\left\{|\hat{f}_u - f_u| \ge \varepsilon f_u\right\} \le \delta & \text{if } f_u \ge m/b \\ \mathbb{P}\left\{|\hat{f}_u - f_u| \ge \varepsilon(m - f_u)\right\} \le \delta & \text{elsewhere} \end{cases}$$

using $O\left((b + (\log(1/\delta)/\varepsilon)(b + 1/\varepsilon))\,(\log m + \log n)\right)$ *bits of space.*

Proof: We suppose that $f_u > 0$, for any u in the input stream (which corresponds to the cash-register data stream model [7]).

Case 1 – Suppose that $\forall v \in N, f_v \le \frac{\varepsilon^2 m}{2k}$. Then, we have also $\forall v \in N, f_v \le m/b$, thus, Line 17 of Algorithm 2 is triggered. Moreover, every counter $C[i][h_i(u)]$ is equal to the sum of the exact frequencies of all data items that collide with u, that is, that share the same hashed value as u (*i.e.*, all $v \in N$ such that $h_i(v) = h_i(u)$). We denote by $X_{u,i}$ the random variable that measures the value of $C[i][h_i(u)] \cdot k/F_0$. We have $X_{u,i} = k/F_0 \sum_{v \in N} f_u \mathbf{1}_{\{h_i(u)=h_i(v)\}}$, where notation $\mathbf{1}_{\{A\}}$ denotes the indicator function, which is equal to 1 if condition A is true and 0 otherwise. By the 2-universality property of the family from which hash function h_i is drawn, we have $\mathbb{E}[h_i(v) = h_i(u)] \le 1/k$, for any v. Thus, by linearity of the expectation,

$$\mathbb{E}[X_{u,i}] = \frac{k}{F_0} \sum_{v \in N} \mathbb{E}\left[f_v \mathbf{1}_{\{h_i(u)=h_i(v)\}}\right] \le \frac{k}{F_0}\frac{m}{k} = \frac{m}{F_0}.$$

Moreover, we have

$$\begin{aligned} \mathrm{Var}[X_{u,i}] &= \mathbb{E}[X_{u,i}^2] - \mathbb{E}[X_{u,i}]^2 \\ &= \frac{k^2}{F_0^2}\left(\sum_{v \in N} f_v^2 \mathbb{E}\left[\mathbf{1}^2_{\{h_i(u)=h_i(v)\}}\right]\right) - \frac{m^2}{F_0^2} \\ &= \frac{1}{F_0^2}\left(k\left(\sum_{v \in N} f_v^2\right) - m^2\right). \end{aligned}$$

Given that $\forall v \in N, f_v \le \frac{\varepsilon^2 m}{2k}$, it is easily checked that $\max \sum_{v \in N} f_v^2 \le \frac{\varepsilon^2 m^2}{2k} + F_0$ and that

$$\mathrm{Var}[X_{u,i}] \le \frac{1}{F_0^2}\left(k\frac{\varepsilon^2 m^2}{2k} - m^2 + kF_0\right) \le \mathbb{E}[X_{u,i}]^2 \frac{k\varepsilon^2}{2k}.$$

Using Chebyshev's inequality, we obtain

$$\mathbb{P}\left\{|X_{u,i} - \mathbb{E}[X_{u,i}]| \ge \varepsilon \mathbb{E}[X_{u,i}]\right\} \le \frac{\mathrm{Var}[X_{u,i}]}{\varepsilon^2 \mathbb{E}[X_{u,i}]^2} \le \frac{1}{2}.$$

This relation holds for any $i \in \{1, \ldots, t\}$ estimators. By using t such estimators, mutually independent, we can estimate the excess of CASE as the minimum of $X_{u,i}$ over all $i \in \{1, \ldots, t\}$. Then, we obtain

$$\begin{aligned} \mathbb{P}\left\{|\hat{f}_u - f_u| \ge \varepsilon f_u\right\} &\le \mathbb{P}\left\{\min_{i \in \{1,\ldots,t\}} \left|X_{u,i} - \frac{m}{F_0}\right| \ge \varepsilon \frac{m}{F_0}\right\} \\ &= \prod_{i \in \{1,\ldots,t\}} \mathbb{P}\left\{|X_{u,i} - \mathbb{E}[X_{u,i}]| \ge \varepsilon \mathbb{E}[X_{u,i}]\right\} \le \frac{1}{2^t} \le \delta \end{aligned}$$

concluding the first case.

Case 2 – Suppose that $f_u \ge m/b$. We have $m - f_u \le m - \frac{m}{b} \le f_u(b-1)$. Thus, we obtain

$$\mathbb{P}\left\{|\hat{f}_u - f_u| \ge \varepsilon f_u\right\} \le \mathbb{P}\left\{|\hat{f}_u - f_u| \ge \varepsilon \frac{m - f_u}{b-1}\right\}.$$

Let us first analyse the excess of a given counter $C[i][h_i(u)]$, $i \in \{1, \ldots, t\}$. We denote by $Y_{u,i}$ the random variable that measures this specific excess. We have

$$Y_{u,i} = \sum_{v \in N \setminus \{u\}} f_v \mathbf{1}_{\{h_i(v)=h_i(u)\}}.$$

Note that $X_u = f_u + \min_{i \in \{1,\ldots,t\}} Y_{u,i}$. Again, by the 2-universality property and by linearity of the expectation, we get that

$$\mathbb{E}[Y_{u,i}] = \sum_{v \in N \setminus \{u\}} \mathbb{E}\left[f_v \mathbf{1}_{\{h_i(v)=h_i(u)\}}\right] \le \frac{m - f_u}{k}.$$

Since for every u in the stream, $f_u \ge 1$ and $Y_{u,i} \ge 0$ for any $i \in \{1, \ldots, t\}$, we can apply the Markov's inequality. Moreover, as $k = \lceil \frac{2(b-1)}{\varepsilon} \rceil$, we get

$$\begin{aligned} \mathbb{P}\left\{Y_{u,i} \ge \varepsilon f_u\right\} &\le \mathbb{P}\left\{Y_{u,i} \ge \varepsilon \frac{m - f_u}{b-1}\right\} \\ &\le \frac{\mathbb{E}[Y_{u,i}](b-1)}{\varepsilon(m - f_u)} \\ &\le \frac{b-1}{k\varepsilon} \le \frac{1}{2}. \end{aligned} \tag{1}$$

This finally leads to

$$\begin{aligned} \mathbb{P}\left\{\hat{f}_u - f_u \ge \varepsilon f_u\right\} &= \mathbb{P}\left\{\min_{i \in \{1,\ldots,t\}} Y_{u,i} \ge \varepsilon f_u\right\} \\ &= \prod_{i \in \{1,\ldots,t\}} \mathbb{P}\left\{Y_{u,i} \ge \varepsilon f_u\right\} \le \frac{1}{2^t} \le \delta. \end{aligned}$$

Finally, the former evaluation of CM algorithm in [2] gives us the last case of the theorem, which concludes the proof. ∎

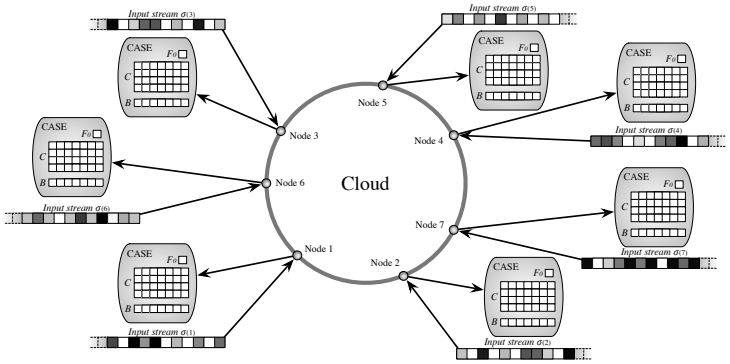


Figure 1: Distributed algorithm structure, where each node runs a CASE instance over a DHT.

IV. A DISTRIBUTED SKETCH FOR FREQUENCY ESTIMATION

In the previous section, we have presented CASE, a sketch that gives for, any ε and any δ, an (ε, δ)-approximation of item frequencies, for any item received in an input stream. We are now ready to present a distributed algorithm that allows to estimate the global frequency of items that appear in distributed streams. This algorithm combines the features of both the streaming model and large scale distributed overlays. As in the streaming model, the input is read on the fly and processed with a minimum workspace and time. As in large scale overlays, the load of the system is partitioned among all the nodes of a cloud system. Specifically, all the nodes of the system are self-organized in a structured overlay. Structured overlays, also called Distributed Hash Tables (DHTs), build their topology according to structured graphs. For most of them, the following principles hold: The identifier space is partitioned among all the nodes of the overlay. Nodes self-organize within the graph according to a distance function d based on nodes identifiers (*e.g.*, two nodes are neighbors if their identifiers share some common prefix), plus possibly other criteria such as geographical distance. Each application-specific object, or data-item, is assigned a unique identifier, called *key*, which is the hashed value of the object. In our context, we consider that this hash function is picked among the 2-universal family. Both keys and node identifiers belong to the same identifier space. Each node owns a fraction of all the data items of the system. The mapping derives from the distance function d. All the proposed DHTs have been proven to be highly satisfactory in terms of both efficiency and scalability (their key-based routing mechanism guarantees operations whose complexity in messages and latency logarithmically scale with system size).

These principles directly apply to our context. All the nodes of $\mathcal{S}$ self-organize in a DHT (see Figure 1), and are responsible for the frequency estimation of all the items whose key are closer to them according to the distance d implemented in the DHT. Let $\mathcal{I}_i$ be the set of items whose keys are closer to node i than to any other nodes, and $\sigma^{(i)}$ denote the input stream of node i. Then, each node $i \in \mathcal{S}$ locally maintains a CASE sketch that solely estimates the frequency of all the items that belong to $\mathcal{I}_i$. For all the other items v that appear in $\sigma^{(i)}$ but that do not belong to $\mathcal{I}_i$, node i routes them to node j such that $v \in \mathcal{I}_j$. Finally, a query for getting the frequency of any item v simply consists in sending this query to the (single) node in charge of v. As will be shown in Theorem 5, this construction guarantees the (ε, δ)-approximation of items frequency, and is space-efficient. Indeed, the total workspace needed by the distributed algorithm (that is the sum of each CASE space implemented at each node $i \in \mathcal{S}$) is strictly equal to the workspace that would be needed by a single CASE to estimate the frequency of all the items recurring in the union of all the distributed streams. Actually, our approach allows us to do statistics over distributed streams without bringing any additional cost with respect to a centralized approach in terms of space however incurs some communication costs. Indeed, each time a node i receives in its input stream an item v that does not belong to $\mathcal{I}_i$, item v must be forwarded to the node j such that j is closer to v key. Note that in expectation this is achieved in a logarithmic number of hops in the size of $\mathcal{S}$. In the following, $m = \sum_{i \in \mathcal{S}} m^{(i)}$. Recall that n is the size of the domain from which data items belong to.

Theorem 5 *The distributed algorithm executed on $s = |\mathcal{S}|$ nodes, each one running an instance of CASE with parameters $k = \lceil e/(s\varepsilon) \rceil$, $t = \lceil \log(1/\delta) \rceil$ and $b < 4/\varepsilon^3$, returns for any item v an estimate $\hat{f}_v$ such that the error bounds of Theorem 4 hold using $O\left((b + (\log(1/\delta)/\varepsilon)(b + 1/\varepsilon))\,(\log m + \log n)/s\right)$ bits of space per node.*

Proof: Let h_{DHT} be the hash function used for the key assignment in the DHT. By the properties of hash functions, the key space can be split into $s = |\mathcal{S}|$ mutually exclusive subsets forming a partition of N. Suppose that all the CASE arrays, spread over the s nodes, are concatenated into a huge unique array whose size is equal to $\lceil \log(1/\delta) \rceil \times \lceil e/\varepsilon \rceil$. The probability that any two items u and v share the same cell on each line is equal to the probability that both u and v are assigned to the same node (which is lower than $1/s$ by the 2-universality properties of h_{DHT}) and that they share the same cell on this node (which is lower than $1/k$). By assumption, the t hash functions are pairwise independent, and by construction, $k = \lceil e/(s\varepsilon) \rceil$. Thus this probability is lower than or equal to ε/e, which corresponds to the probability obtained with a unique CASE instance with parameter $k = \lceil e/\varepsilon \rceil$ and $t = \lceil \log(1/\delta) \rceil$.

We now focus on the Space Saving algorithm run on each node in $\mathcal{S}$ with parameter $b \leq 4/\varepsilon^3$. Each node in charge of n/s items receives a stream of size m/s in average. Thus according to Theorem 2, for every v in the any stream, we have

$$f_v \geq \hat{f}_v^{(\mathtt{SS})} \geq f_v - \frac{m/s}{b} > f_v - \frac{m}{b}.$$

Thus the guarantee brought by the union of all instances of the enhanced Space Saving on each node in $\mathcal{S}$ is strictly greater to the one guaranteed by a single CASE instance with the same parameter $b \leq 4/\varepsilon^3$.

Direct application of Theorem 4 concludes the proof. ■

V. EFFORT NEEDED BY THE ADVERSARY TO SUBVERT THE DISTRIBUTED ALGORITHM

As previously said, we suppose that the adversary has enough resources to generate a large number of items, and to judiciously inject them in the input stream $\sigma^{(i)}$ of any correct node $i \in \mathcal{S}$, so that items frequency are over-estimated.

From Algorithm 2, this can be only achieved by increasing the error made on the estimations $\hat{f}_v^{(\mathtt{CM})}$ of item v as by Theorem 2 we have that $f_v - m/b \leq \hat{f}_v^{(\mathtt{SS})}$ holds. Thus to disrupt the estimation

Table I: Key values of E_k

k	10 ($\varepsilon \sim 0.3$)		50 ($\varepsilon \sim 0.05$)		250 ($\varepsilon \sim 0.01$)	
η	10^{-1}	10^{-4}	10^{-1}	10^{-4}	10^{-1}	10^{-4}
E_k	44	110	306	651	$1,617$	$3,363$

$\hat{f}_v^{(\mathtt{CM})}$ of any item v, the adversary has to generate sufficiently many items $u_1, \ldots, u_\ell$ such that for all the lines q, $q = 1, \ldots, t$ of array C, there exists an item u_j such that $h_q(u_j) = h_q(v)$. Recall that the t hash functions are locally chosen, thus the adversary cannot know which identifiers map to $h_1(v), \ldots, h_t(v)$. By injecting numerous times these items $u_1, \ldots, u_\ell$, the estimation $\hat{f}_v^{(\mathtt{CM})}$ will be arbitrarily overestimated. Note that the adversary will blindly bias the frequency estimation of many items, including its owns.

By conducting an analysis similar to the one achieved in [16], we can derive the minimum effort that needs to be exerted by the adversary to make its attack successful with probability $1-\eta$, where $\eta < 1$. This is achieved by modeling an attack as a urn problem, where each entry of C is modeled as an urn and each received distinct item as a ball. N_ℓ is the random variable that counts the number of non empty urns among any set of k urns at time ℓ. Let U_k be the number of balls needed in order to obtain all the k urns occupied, *i.e.*, with at least one ball. It is easily checked that $\mathbb{P}\{U_1 = 1\} = 1$ and that, for $\ell \geq k \geq 2$, we have

$$U_k = \ell \Longrightarrow N_{\ell-1} = k - 1.$$

From [16], we get, for $k \geq 2$ and $\ell \geq k$,

$$\mathbb{P}\{U_k = \ell\} = \frac{1}{k^{\ell-1}} \sum_{r=0}^{k-1} (-1)^r \binom{k-1}{r} (k-1-r)^{\ell-1}.$$

Finally, we consider the integer E_k which counts the number of balls needed to get a collision in all the $k \times t$ urns. Note that this number is independent of t as by definition, the t experiments in parallel are identical and independent. Thus, filling entirely a set of k urns leads to obtain all the t sets of k urns occupied. For given value of k and $\eta \in (0,1)$, integer E_k is defined by

$$E_k = \inf \left\{ \ell \geq k \,\middle|\, \sum_{i=k}^{\ell} \mathbb{P}\{U_k = i\} > 1 - \eta \right\}. \tag{2}$$

Recall that parameters k of Algorithm 2 are common knowledge (except the random local coins) and thus the adversary is capable of deriving E_k according to the desired probability η. Finally, the adversary needs to inject in the input stream $E_k + a$ items to bias the frequency estimation of all the items that have been received in the input streams.

The main results of this analysis are summarized in Table I. The most important result is that the effort that needs to be exerted by the adversary to subvert the estimation can be made arbitrarily large by any correct node by just increasing the size of the array. This also entails that the adversary effort is not related with the population size.

Table II: Dataset Statistics.

Datasets	stream size (m)	distinct items (F_0)	max. freq.
NASA (Jul.)	1,891,715	81,983	17,572
NASA (Aug.)	1,569,898	75,058	6,530
ClarkNet (Aug.)	1,654,929	90,516	6,075
ClarkNet (Sep.)	1,673,794	94,787	7,239
Saskatchewan	2,408,625	162,523	52,695

VI. Performance Evaluation of CASE

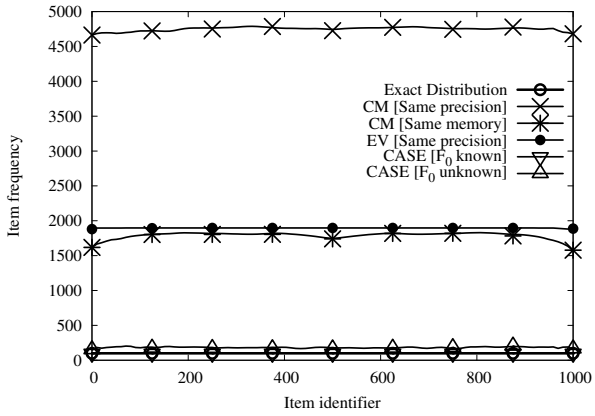
This section describe the main results obtained from the experiments evaluating the quality of CASE to estimate on the fly the frequency of a very large number of items in a massive stream.

A. Experimental Setup

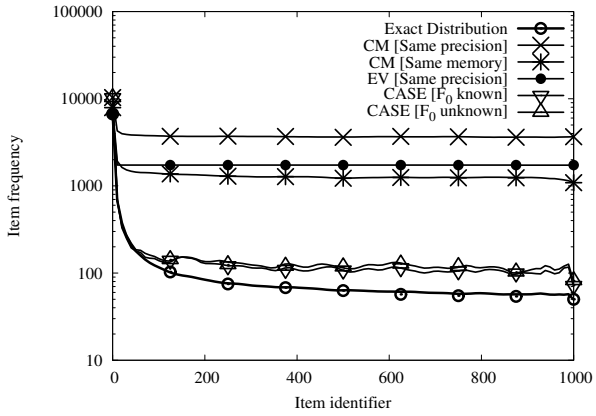
We have implemented CASE and run a series of experiments on different types of streams and for different parameter settings. We have fed our algorithm with both real-world data sets and synthetic traces. Real data give a realistic representation of some existing systems, while the latter ones allow us to capture phenomenon which may be difficult to obtain from real-world datasets, and thus allow us to check the robustness of our strategies. We have varied all the significant parameters of the algorithms, that is, the size of the Space Saving algorithm memory b, the number of entries k in each line of the CM matrix, and the number t of lines of this matrix. We have then compared the frequency estimation provided by CASE with the one provided by CM algorithm. For each parameter setting, we have conducted and averaged 100 trials of the same experiment, leading to a total of more than $1,000,000$ experiments for the comparison of the Count-Min algorithm and ours. The datasets have been downloaded from the repository of Internet network traffic [17]. We have used three large datasets among the available ones, representing different periods of HTTP requests to WWW server of respectively NASA Kennedy Space Center, ClarkNet provider and University of Saskatchewan. Table II shows the stream size (m), the population size (F_0) and the number of occurrences of the most frequent item of these datasets. Note that all these benchmarks share a Zipfian behavior, with a lower α parameter for the University of Saskatchwan. For more information on these datasets, an extensive analysis is available in [18].

B. Main Lessons drawn from the experiments

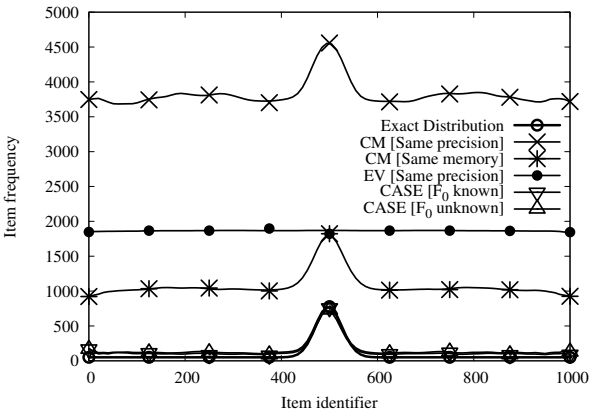
We now present the main lessons drawn from these experiments. Due to space constraints, only a subset of all the conducted experiments are included in this paper. In order to compare the accuracy of the presented algorithms, we measure the distance between the frequency of the items in the input stream and the one computed by the CM algorithm and the CASE algorithm. The distance we use is the *euclidean distance*. Specifically, given two vectors of dimension n, v and w, the euclidean distance is defined by $\mathcal{D}(v,w) = \sqrt{\sum_{i=1}^{n}(v_i - w_i)^2}$. In the following, we denote with "CM [Same precision]" the results achieved by CM when initialized with the same ε and δ values of CASE. On the other hand, "CM [Same Memory]" denotes the results achieved by CM when initialized with values of ε and δ such that the CM has the same memory usage (number of bits) of CASE (*i.e.*, the CM matrix is larger). To identify the impact of F_0 estimation on the quality of



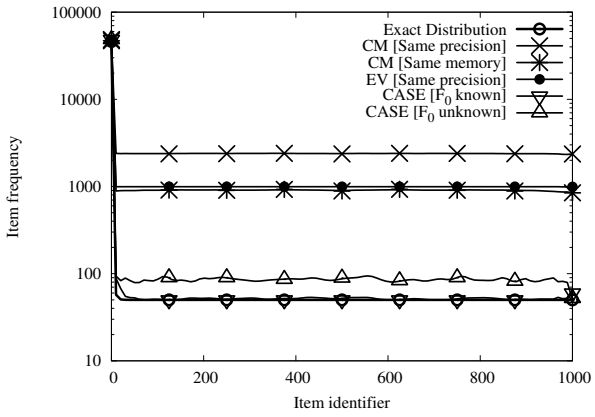
(a) Uniform distribution



(b) Zipfian-like distribution – $\alpha = 1$



(c) Truncated Poisson distribution



(d) Zipfian-like distribution – $\alpha = 4$

Figure 2: Item frequency as a function of their identifiers. Settings: $m = 100,000$; $F_0 = 1,000$; $\varepsilon = 0.1$; $\delta = 0.1$.

CASE, we provide the results both when CASE knows the exact value of F_0, denoted as "CASE [F_0 known]", and when it relies on the estimation of F_0 provided by the BJKST algorithm, denoted as 'CASE [F_0 unknown]". Finally, "EV [Same precision]" denotes results obtained with the algorithm proposed in [4], providing a comparison with respect to related work, when initialized with the same ε and δ values of CASE.

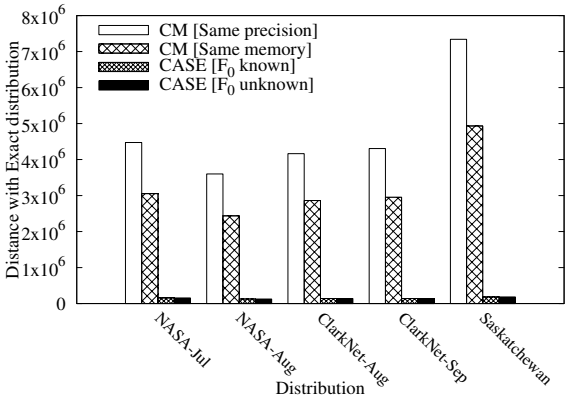
Figure 2 compares, for different input stream shapes, the frequency estimation provided by CM and CASE algorithms. The main observation drawn from this figure is that CASE clearly outperforms CM algorithm. In accordance with the analysis, in presence of an input stream uniformly distributed (Figure 2(a)) or Poisson-distributed (Figure 2(c)), CASE perfectly estimates item frequency in contrast to CM. Indeed, in both scenarios, all the items (or most of them in the Poisson case) show the same frequency which allows CASE to exhibit its optimum behaviour. Notice also that providing more space to CM still does not suffice to reach CASE accuracy. Now for highly skewed distributions (Figures 2(b) and 2(d)), CASE overestimates a little bit the frequency of rare items (by a factor 1.2), while CM, in the best case, overestimates them by a factor 100. Note that in both cases, the frequency of rare items are lower-bounded by 50, in order to emphasize the precision of CASE estimation (indeed, as shown in Theorem 4, the precision of both CASE and CM is similar for strongly rare items). Notice that as expected both CM and CASE perfectly estimate highly frequent items.

On the other hand, Figure 3(a) compares the quality of both CM and CASE to estimate item frequency when they are fed with massive real datasets. This figure confirms that CASE clearly outperforms CM estimation. Indeed, the error made by CM is function of the size of the stream, which is not the case for CASE except for rare items.

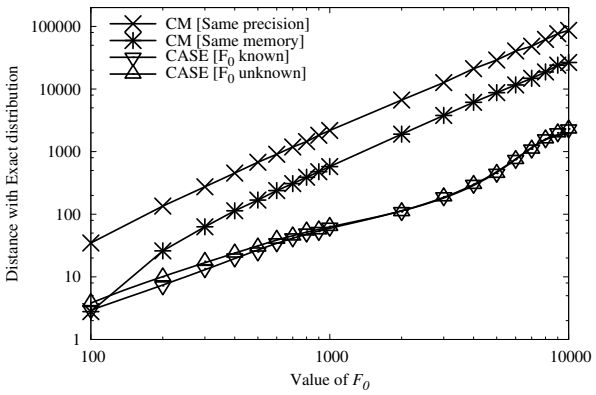
We now compare the behavior of both algorithms by focusing on their parameters. Figure 3(b) shows the euclidean distance between the exact frequency distribution and the one estimated by CM and CASE. When the number of distinct items is close to k (the number of columns of the matrix) both algorithms output correct estimations since almost all the distinct items have their own cell in the matrix. On the other hand, for very large values of F_0, the distance between the exact frequency distribution and the estimated one does augment because the number of collisions is extremely high, even for CASE. Figure 3(c) confirms the results obtained with real datasets, that is, that CASE outperforms CM in presence of very large input streams. Finally, Figure 3(d) illustrates the intuitive idea that augmenting the memory space used by the sketch increases the precision of estimation. This figure clearly confirms the theoretical analysis.

VII. Conclusion

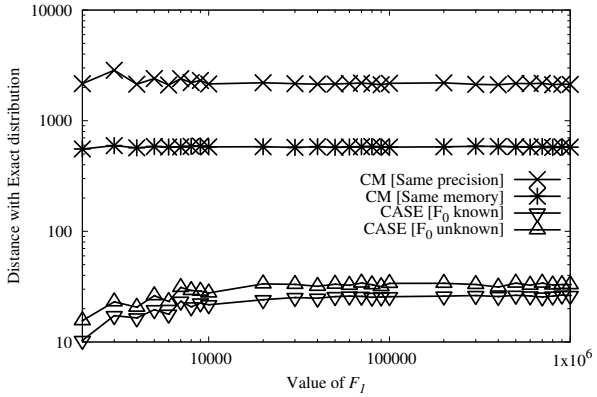
In this paper we have investigated the problem of estimating on the fly the frequency at which items recur in massive distributed data



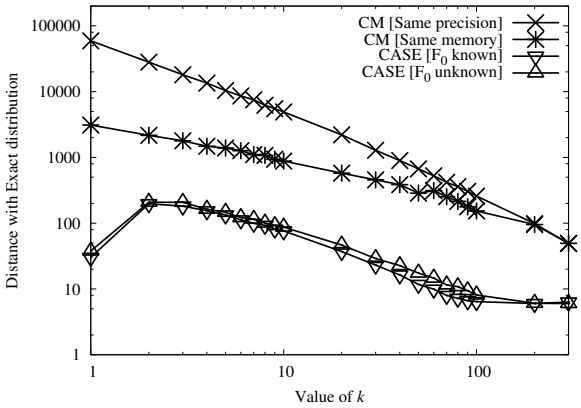
(a) CM and CASE are fed with real datasets
Setting: $\varepsilon = 0.01$ and $\delta = 0.1$



(b) ED as a function of the number of distinct items
Setting: $m = 100,000$; $\varepsilon = 0.1$; $\delta = 0.1$.



(c) ED as a function of the size of the stream
Setting: $F_0 = 1,000$; $\varepsilon = 0.1$; $\delta = 0.1$.



(d) ED as a function of the Count-Min sketch parameter k
Setting: $m = 100,000$; $F_0 = 1,000$; $\delta = 0.1$

Figure 3: Euclidean Distance (ED) between the frequency of the items when generated by real datasets (case (a)) and by a Poisson distribution (cases (b), (c) and (d)) and the frequency of the items estimated by the Count-Min sketch and the CASE algorithm.

streams. Our approach, denoted CASE algorithm, combines tools and probabilistic algorithms from the data streaming model. By doing so, we have improved upon the Count-Min sketch algorithm by providing an algorithm that guarantees an relative (ε, δ)-approximation of item frequency estimation, provided these items are not too rare. We have also provided a distributed algorithm that allows us to globally determine the frequency of all the items that recur in these streams. Experiments on both synthetic and real datasets validate our theoretical analysis.

REFERENCES

[1] E. Anceaume, Y. Busnel, and S. Gambs, "On the Power of the Adversary to Solve the Node Sampling Problem," *Transactions on Large-Scale Data- and Knowledge-Centered Systems (TLDKS)*, vol. 8290, no. 11, pp. 102–126, 2013.

[2] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[3] X. Dimitropoulos, M. Stoecklin, P. Hurley, and A. Kind, "The eternal sunshine of the sketch data structure," *Computer Networks*, vol. 52, no. 17, 2008.

[4] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.

[5] E. D. Demaine, R. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *In Proceedings of the 10th Annual European Symposium on Algorithms*. Springer-Verlag, 2002, pp. 348–360.

[6] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[7] Muthukrishnan, *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005.

[8] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[9] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. of the 28th annual ACM symposium on Theory of computing (STOC)*, 1996, pp. 20–29.

[10] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Proc. of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*. Springer-Verlag, 2002, pp. 1–10.

[11] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the 10th International Conference on Database Theory*, ser. ICDT'05. Springer-Verlag, 2005, pp. 398–412.

[12] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[13] P. Gibbons, *Data Streams Management: Processing High-Speed Data Streams*. Elsevier, 2007.

[14] P. B. Gibbons and S. Tirthapura, "Estimating simple functions on the union of data streams," in *Proc. of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001, pp. 281–291.

[15] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, 1982.

[16] E. Anceaume, Y. Busnel, and B. Sericola, "Uniform node sampling service robust against collusions of malicious nodes," in *Proc. of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Budapest, Hungary, June 2013.

[17] the Internet Traffic Archive, "http://ita.ee.lbl.gov/html/traces.html," Lawrence Berkeley National Laboratory, Apr. 2008.
[18] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: the search for invariants," *SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.