Distinct Random Sampling from a Distributed Stream

Yung-Yu Chung and Srikanta Tirthapura
2215 Coover Hall, Iowa State University, Ames, IA, USA
$\{$*ychung,snt*$\}$*@iastate.edu*

***Abstract*—We consider continuous maintenance of a random sample of distinct elements from a massive data stream, whose input elements are observed at multiple distributed sites that communicate via a central coordinator. At any point, when a query is received at the coordinator, it responds with a random sample from the set of all distinct elements observed at the different sites so far.**

We present the first algorithms for distinct random sampling from a distributed stream. We also present a lower bound on the expected number of messages that must be transmitted by any distributed algorithm, showing that our algorithm is message optimal to within a factor of four. We present extensions to sliding windows, and experimental results showing the performance of our algorithm on real-world data sets.

***Keywords*-Distinct Sampling, Random Sampling, Distributed Stream Mining, Distinct Elements**

I. Introduction

A random sample is a flexible and general synopsis of data, and can be used to answer aggregate queries on data with provable probabilistic guarantees on the quality of the answer. A *distinct random sample* (commonly known as a "distinct sample") is chosen from the set of all distinct elements in data. A distinct sample has the property that the probability of an element's inclusion in the sample is independent of the frequency of the element, as long as it occurs within the data. This property is useful in computing an important class of aggregates that depend on the set of distinct elements in data. For instance, a distinct sample can be used in answering the queries: "what is the number of distinct IP addresses observed within a stream of IP packets during a given hour", or "how many distinct visitors have used a web service, who come from a particular country?", or "what is the average age of the distinct users accessing this website"? In contrast with a distinct sample, a *simple random sample* is chosen from the set of all occurrences within the data. The greater the frequency of an element in data, the more likely its inclusion within a simple random sample. A simple random sample is not suited to estimate answers to the above queries, while a distinct sample is.

A simple random sample favours the "heavy-hitters" within data, while a distinct sample does not. In cases where the distribution of elements is skewed, i.e. a few distinct elements account for a majority of the data elements, a simple random sample will likely ignore elements with low frequency and only contain elements of high frequency (the "heavy hitters"). In contrast, a distinct random sample is equally likely to contain low frequency as well as high frequency elements. Due to this property, a distinct sample of a data stream has found applications in database query processing [1], network monitoring [2], and in detecting anomalous network traffic [3].

Distinct sampling is a fundamental primitive in current day "big data" systems. Query optimizers in commercial relational database systems maintain a distinct sample of individual database columns to help in choosing efficient query processing strategies [4]. In stream database systems, sampling plays a fundamental role as a data filtering method to reduce processing load, and typically a variety of sampling methods are implemented, including distinct sampling [5]. Systems such as BlinkDB [6] use sampling to process queries quickly, in exchange for approximate accuracy guarantees, and have been successful in providing dramatic speedups for common aggregate queries. These systems operate on tens of terabytes of data, spread over hundreds of machines [6], and distributed sampling is a key component of their query execution strategy.

We consider the maintenance of a distinct sample from a stream whose elements are observed at multiple distributed sites that communicate via a network. We present a memory and communication-efficient solution suited to today's stream processing frameworks such as IBM Infosphere Streams [7], Apache Spark [8] and Apache Storm [9]. For the purpose of describing and analyzing the algorithm, we work in the *continuous distributed monitoring* model [10]–[13], where a coordinator node is required to continuously maintain an aggregate function over the union of all streams observed at the different distributed sites. For distributed processing of large streams, the bottleneck is often the network, rather than processor speed, and hence we focus on minimizing the message complexity of the streaming algorithm.

Sampling from a stream has a long history of research, starting from the popular *reservoir sampling* algorithm due to Waterman (see Algorithm R from [14]) that has been known since the 1960s. This includes work on speeding up reservoir sampling [14], weighted reservoir sampling [15], sampling over a sliding window and stream evolution [16]–[19], and distributed random sampling [10], [12]. While there is prior work on distinct sampling from a single stream [1], [20]–[22], these do not extend to continuous distributed monitoring.

A. *Contributions*

We make the following contributions. Let k denote the number of distributed sites, s the desired sample size, and d the total number of distinct elements in the entire distributed stream.

- We present a distributed algorithm for continuously maintaining a distinct sample from a distributed stream whose expected message complexity is $O\left(ks \ln \frac{d}{s}\right)$. When a query is posed at the central coordinator, it responds with a distinct sample of size $\min\{s, d\}$ from the set of all elements seen so far in the stream.
- We present a matching lower bound showing that *any* algorithm for continuously maintaining a distinct sample from the distributed stream must have a message complexity of $\Omega\left(ks \ln \frac{d}{s}\right)$. In other words, the message complexity of our algorithm is optimal to within a constant factor.
- We present an extension to support queries over a sliding window of the distributed data stream. We show this algorithm is also near-optimal in its message complexity through demonstrating an appropriate lower bound.
- We implemented our algorithm and evaluated its performance over real-world datasets. We found that our algorithms are easy to implement and provide a message complexity that is typically much better than the worst case bounds that we derive.

Comparison with Simple Random Sampling: It is interesting to compare the message complexity of distributed distinct sampling (DDS) to that of distributed simple random sampling (DRS). From previous work [10], [12], it is known that the message complexity of DRS for a sample size s for n total elements distributed over k sites increases as approximately $\max\{k, s\} \log(n/s)$ [1] In contrast, the message cost of DDS increases as $k \cdot s \cdot \log(d/s)$, where k is the number of sites, d the total number of distinct elements, and s the sample size. The message complexity of DDS is inherently larger than that of DRS if d (the number of distinct elements in the stream) is comparable to n (the total number of elements in the stream).

To understand the reason for this difference, note that in DRS, when n elements have already been received in the system, the probability of a new element being selected into the sample decreases as s/n. In case of DDS, the probability of a new element being selected into the sample decreases as s/d, where d is the number of distinct elements received by the system. Due to the possibility of the same element appearing at different sites at a similar time, many sites may be “fooled” into sending messages to the coordinator after simultaneously observing the same element, and hence more messages may be communicated between the sites and the coordinator in DDS. Our lower bound shows that this is not an artifact of an inferior algorithm for DDS. Instead, greater coordination is inherently necessary for DDS than for DRS.

[1]The exact expression is $\Theta\left(\frac{k \log(n/s)}{\log(k/s)}\right)$ if $s < k/8$ and $\Theta(s \log(n/s))$ if $s \geq k/8$. This message complexity is tight up to a constant factor, due to matching lower bounds.

B. *Prior and Related Work*

There is a growing literature on algorithms for continuous distributed monitoring, starting from the basic “countdown problem” [23], frequency moments [11], [23], entropy [24], heavy-hitters and quantiles [25], geometric methods for monitoring [26], [27], and often, matching lower bounds [11]. For a recent survey on results in this model, see [13]. A survey of sampling on streams appears in [28].

The closest prior work is the work on algorithms and lower bounds for distributed random sampling [10], [12]. While the problems are different, our algorithm for DDS on an infinite window is essentially an adaptation of the DRS algorithm from [12] with a few important differences: one is that our algorithm uses sampling based on a randomly chosen hash function while the algorithm for DRS samples based on independent random bits. Further, our algorithm employs additional data structures to prevent duplicate messages from a site to the coordinator due to observing the same element, this is not necessary for simple random sampling. The analysis of the message complexity is different for the DRS and the DDS algorithms.

The lower bound arguments for the two problems (DRS and DDS) are very different from each other. Our analysis of the lower bound on message cost of any algorithm for DDS uses methods, that to our knowledge, have not been applied to the continuous distributed streaming model.

A related model of distributed streams was considered in [20], [29]. In this model, the coordinator was not required to continuously maintain an aggregate, but instead, when the query was posed to the coordinator, the sites would be contacted and the query result would be constructed. In their model, the coordinator could be said to be “reactive”, whereas in the model considered in this paper (the continuous distributed streaming model), the coordinator is “pro-active”.

Roadmap. We first present the model and problem definition in Section II, the algorithm and lower bound for infinite window in Section III, an overview of the algorithm for sliding window in Section IV, and experimental results in Section V. Due to space constraints, we omit some of the proofs, and also omit details of the sliding windows algorithms and experimental results. These details, including all the proofs, can be found in the full version of the paper [30].

II. Model

We consider a system with k sites, numbered from 1 to k. Each site i monitors a local stream of elements, which are not all necessarily distinct. There is an integer time associated with each observation, and time is non-decreasing within a stream. At time t, let $\mathcal{S}_i(t)$ denote the stream observed by site i so far, and let $\mathcal{S}(t) = \cup_{i=1}^{k} \mathcal{S}_i(t)$ denote the stream observed by the system so far. Let $\mathcal{D}(t)$ denote the set of distinct elements in $\mathcal{S}(t)$, $n(t)$ the number of elements in $\mathcal{S}(t)$, and $d(t)$ the number of distinct elements in $\mathcal{S}(t)$.

There is a coordinator node, different from the other k sites, to whom all queries are posed. The different sites as well as the coordinator are assumed to be synchronized in time, and we assume that the message delay from the sites to the coordinator are all equal, as in a synchronous distributed system. These assumptions, which have also been used in previous works in the continuous distributed streaming model [12], [23], [31], allow us to focus on communication efficiency.

We consider two versions of distinct sampling, based on the scope of the data that the query addresses. In the *infinite window* case, at every time instant t, the coordinator must maintain a random sample of size $\min\{s, d\}$ from $\mathcal{D}(t)$. In the *sliding window* case [32], we are given a window size w as a parameter, and the user is interested in all elements that have arrived in the most recent w time intervals. In particular, let $\mathcal{S}_i^w(t)$ denote the stream that has arrived at site i at times $t-w+1, t-w+2, \ldots, t$, and let $\mathcal{S}^w(t) = \cup_{i=1}^{k} \mathcal{S}_i(t)$. Let $\mathcal{D}^w(t)$ denote the set of distinct elements in in $\mathcal{S}^w(t)$. The goal is for the coordinator to maintain at all times t, a random sample of size $\min\{s, d\}$ from the elements in $\mathcal{D}^w(t)$.

Our measure of performance is the total number of messages sent between the coordinator and the sites, and we aim to minimize this number. Since each message in our algorithm is of a small size, this is also a measure of the number of bytes transmitted, in our case. The size [2] of local data stream S_i, the order of arrival, the number of distinct elements in the local stream, and the interleaving of the stream at different sites, can all be arbitrary, and the algorithm cannot make any assumption about these.

III. Infinite Window

We first consider the case of infinite window, when the sample has to be chosen from the set of all distinct elements seen so far in the stream. Let $h : \mathcal{U} \rightarrow [0, 1]$ be a hash function that assigns a real number in the range $[0, 1]$ to each element in $\mathcal{U}$. For different inputs, it is assumed that the outputs of h are mutually independent random variables. For set S, let $h(S)$ denote $\{h(x) | x \in S\}$. Note that for any time t, $h(\mathcal{S}(t)) = h(\mathcal{D}(\mathcal{S}(t)))$. The basic sampling strategy is as follows. *The distinct sample at time t is the set of elements from $\mathcal{S}(t)$ that yield the s smallest elements in $h(\mathcal{S}(t))$.*

It is clear that the above yields a distinct sample from $\mathcal{S}(t)$. To see this, take any subset $T \subseteq \mathcal{D}(\mathcal{S}(t))$ of size s; the probability that the elements in T will yield the s smallest values in $h(\mathcal{D}(\mathcal{S}(t)))$ is exactly the probability that in a random permutation of $h(\mathcal{D}(\mathcal{S}(t)))$, the elements in T are ordered before the rest, which is $1/\binom{|\mathcal{D}(t)|}{s}$.

[2]The message size in our algorithm is constant, assuming that each stream element can be stored in a constant number of bytes.

Our distributed algorithm for maintaining the above sample is as follows. Let $u(t)$ denote the value of the sth smallest element in $h(\mathcal{S}(t))$. The coordinator always has the current value of $u(t)$. Each site i maintains a state variable $u_i(t)$, which is its local view of $u(t)$. $u_i(t)$ is initialized to 1 and is updated as follows. Whenever site i observes an item e such that $h(e) < u_i(t)$, e and $h(e)$ are sent to the coordinator, who updates $u(t)$. If $h(e)$ indeed changed the value of $u(t)$, then e is selected into the sample at the coordinator, replacing a current element in the sample (unless fewer than s distinct elements have been seen so far). In turn, the coordinator sends a message back to i to refresh the value of $u_i(t)$. The algorithm at site i is presented in Algorithm 1 and at the coordinator is in Algorithm 2.

Algorithm 1: Infinite Window: Algorithm at Site i

```
Initialization: Receive hash function $h$ from the coordinator; $u_i \leftarrow 1$, $P_i \leftarrow \emptyset$;
repeat
    if receive element $e$ from $\mathcal{S}_i$ then
        if $h(e) < u_i$ and $e \notin P_i$ then
            Insert $e$ into $P_i$ ;
            Send $e$ to the Coordinator;
    if receive element $u$ from the coordinator then
        $u_i \leftarrow u$;
        Discard all elements $e'$ in $P_i$ such that $h(e') > u_i$;
until Forever;
```

Lemma 1. *When queried at time t, the coordinator returns a random sample of size $\min\{s, d\}$ selected without replacement from $\mathcal{D}(t)$.*

Proof: First note that at time t, variable u_i tracks $u_i(t)$ at site i, and u tracks $u(t)$ at the coordinator. For each site i we claim $u_i \geq u$ at all times. This can be proved using induction. For the base case, note that initially, $u_i = u$. There are two types of events to consider for the inductive step: (1) when u us updated, and (2) when u_i is updated. Each time u is updated, it only becomes smaller (i.e. u is non-increasing), which maintains the invariant. Each time u_i is updated, it is set to the current value of u, which also maintains the invariant.

Assuming that hashes of different elements are distinct, we claim that the value of u equals the ℓth smallest hash value seen so far, where $\ell = \min\{s, d\}$. We claim that every

Algorithm 2: Infinite Window: Algorithm at the Coordinator

```
/* $P$ is the random sample, and
   variable $u$ has the current value of
   $u(t)$.                                    */
Initialization: $P \leftarrow \emptyset$, $u \leftarrow 1$;
repeat
    if receive $e$ from site $i$ then
        if $h(e) < u$ then
            If $e \notin P$, then insert $e$ into $P$;
            if $|P| > s$ then
                Discard element
                $e' = argmax\{h(e)|e \in P\}$ from $P$;
                Update $u \leftarrow \max\{h(e)|e \in P\}$;
        Send $u$ to site $i$;
    if a query for a random sample arrives then
        Send $P$;
until Forever;
```

time a new element e arrives at site i such that $h(e) < u$, e will be sent to the coordinator. To see this, consider an element e such that $h(e) < u$, observed at site i. We have $u \leq u_i$, and $h(e) < u$; these together imply that $h(e) < u_i$. From the algorithm at the site, it is clear that e will be sent to the coordinator. Thus, the coordinator can inductively maintain the ℓ elements from $\mathcal{D}(t)$ with the smallest hash values. It is clear this constitutes a random sample of size ℓ chosen without replacement from $\mathcal{D}(t)$. ■

A. *Infinite Window: Analysis*

We present an analysis of the message and space complexities of our algorithm. In Section III-A1 we present an upper bound on the message complexity of our algorithm and in Section III-A2 we present a lower bound.

1) Upper Bound: Consider the execution of the algorithm until the end of time step t. Let $d = |\mathcal{D}(t)|$ be the total number of distinct elements that were observed in the stream. Let d_i denote the total number of distinct elements in $\mathcal{S}_i(t)$. Clearly, $d_i \leq d$. Let Y_i denote the total number of messages that are sent by node i (note that the number of messages sent by site i equals the number of messages received). Let Y denote the total number of messages sent in the system. In the following, when the context is clear, we use $\mathcal{S}_i$ to mean $\mathcal{S}_i(t)$, $\mathcal{D}_i$ to mean $\mathcal{D}_i(t)$, and so on.

Lemma 2. *An element e observed at site i multiple times will not lead to more than one message transmission from i to the coordinator, and this transmission can only happen the first time site i observes e.*

Proof: Suppose that e is observed by site i at times $t_1, t_2, \dots$ where $t_1 < t_2 < \dots$. The first time it is observed at time t_1, suppose that a message was not sent to the coordinator. In this case, $h(e) \geq u_i(t_1)$. Since u_i is non-increasing, when e is observed at a future time, say t_2, it will also be true that $h(e) \geq u_i(t_2)$, and no further messages are sent to the coordinator. Next, suppose that a message was sent to the coordinator when e was observed at time t_1. In this case e is inserted into P_i at time t_1. When e is observed again at time t_2 (or later), either e is still in P_i, or it must be true that $u_i(t_2) < h(e)$, since otherwise, e would not have been discarded from P_i. Hence, e will not be sent to the coordinator in this case, according to Algorithm 1. ■

For $j = 1 \dots d_i$, let e_i^j be the jth new distinct element in the local stream $\mathcal{S}_i$. For $j = 1 \dots d_i$, let Y_i^j be a random variable equal to 1 if site i communicated with the coordinator upon receiving e_i^j, and 0 otherwise. We have $Y_i = \sum_{j=1}^{d_i} Y_i^j$.

Lemma 3. *For site i, and $j = 1 \dots d_i$, $\Pr\left[Y_i^j = 1\right] \leq \frac{s}{j}$, if $j > s$.*

Proof: Let Z_i^j denote the event that $h(e_i^j)$ is among the s smallest elements in $\{h(e_i^q)|q = 1 \dots j\}$. Note that if $Y_i^j = 1$, then Z_i^j must be true. Note that the converse is not necessarily true; it is possible that Z_i^j is true, but $Y_i^j = 0$, for example, e_i^j may have already been observed by a site other than i, and its value may have been incorporated into u, and hence into u_i. It is easy to see that $\Pr\left[Z_i^j\right] = \frac{s}{j}$ for $j > s$. Since $\Pr\left[Y_i^j = 1\right] \leq \Pr\left[Z_i^j\right]$, the lemma follows. ■

Lemma 4.

$$\mathbb{E}\left[Y_i\right] \leq s + s\left(H_{d_i} - H_s\right)$$

Proof: Using linearity of expectation on Y_i, we get:

$$\begin{aligned}
\mathbb{E}\left[Y_i\right] &= \sum_{j=1}^{d_i} \mathbb{E}\left[Y_i^j\right] = \sum_{j=1}^{d_i} \Pr\left[Y_i^j = 1\right] \\
&= \sum_{j=1}^{s} \Pr\left[Y_i^j = 1\right] + \sum_{j=s+1}^{d_i} \Pr\left[Y_i^j = 1\right] \\
&\leq s + \sum_{j=s+1}^{d_i} \frac{s}{j} = s + s\left(H_{d_i} - H_s\right)
\end{aligned}$$

where we have used Lemma 3. ■

Lemma 5. *Let Y denote the number of messages transmitted by the distributed algorithm during an execution when d distinct elements are observed overall.*

$$\mathbb{E}\left[Y\right] \leq 2ks + 2ks\left(H_d - H_s\right) \approx 2ks\left(1 + \ln\left(\frac{d}{s}\right)\right)$$

Proof: Note that for each message sent by a site, there is exactly one message sent by the coordinator, and hence we have $Y = 2\sum_{i=1}^{k} Y_i$. The proof follows from Lemma 4 combined with the observation $d_i \leq d$. ■

We remark that using Lemma 4, it is possible to get a tighter upper bound for $\mathbb{E}\left[Y\right]$ in cases when the numbers of distinct elements observed at individual sites is much smaller than the number of distinct elements overall.

Observation 1.

$$\mathbb{E}\left[Y\right] \leq 2ks + 2s\sum_{i=1}^{k}\left(H_{d_i} - H_s\right) \approx 2ks + 2s\sum_{i=1}^{k}\ln\left(\frac{d_i}{s}\right)$$

The following theorem summarizes the performance of the algorithm for infinite windows.

Theorem 1. *Let d, s, and k respectively denote the total number of distinct elements in the distributed stream, sample size, and the number of sites. There is an algorithm that continuously maintains a distinct sample of a distributed stream, whose expected total number of messages is no more than $2ks\ln\left(\frac{de}{s}\right)$, memory consumption per site is $O(s)$, memory consumption at the coordinator is $O(s)$, and processing time per element is $O(1)$.*

Proof: The message complexity follows from Lemma 5. The memory at the coordinator is also clear from an inspection of Algorithm 2. For the memory at site i, note that as soon as an element e is inserted into P_i, e is also sent to the coordinator, and in turn the site receives the current value of u, sets $u_i \leftarrow u$, and discards all elements e from P_i such that $h(e) > u$. Globally, there are no more than s distinct elements e such that $h(e) \leq u$, hence even within site i, there are no more than s elements in P_i. ■

2) Lower Bound: Given any distributed algorithm $\mathcal{A}$ that continuously maintains a distinct sample of the stream of size s, we construct an input which causes $\mathcal{A}$ to send at least a certain minimum number of messages, in expectation. Suppose that the elements were all chosen from the set $[m] = \{1, 2, \ldots, m\}$ for $m >> k$. Note that the probability space here is the one from which the random sample is chosen; every algorithm needs access to random bits that define this probability space, such that the sample chosen is uniformly chosen from the set of all distinct elements observed so far.

Lemma 6. *Suppose a set $\mathcal{D}$ of distinct elements of size d has already been observed by the system so far, after some rounds of computation. For any algorithm $\mathcal{A}$, and any site $i = 1 \ldots k$, there is an element $e_i^{\mathcal{D}}$ such that upon receiving any element $e \in [m] - e_i^{\mathcal{D}} - \mathcal{D}$ in the next round, site i will send a message to the coordinator with probability at least $\frac{s}{2(d+1)}$.*

Proof: We use proof by contradiction. Note that set $\mathcal{D}$ has already been observed by the system so far. Suppose that there were two distinct elements $e_i, e_i' \in [m] - \mathcal{D}$ such that upon e_i, the probability that i sent a message to the coordinator was less than $\frac{s}{2(d+1)}$, and also upon e_i', the probability that i sent a message to the coordinator was less than $\frac{s}{2(d+1)}$.

Consider the following two inputs in the next round. In one input $\mathcal{I}^1$, site i is given e_i and the other sites do not receive any element. In the other input $\mathcal{I}^2$, site i is given e_i' and the other sites do not receive an element. In $\mathcal{I}^1$, at the end of this round, there is a probability of $\frac{s}{d+1}$ that e_i belongs to the random sample. In $\mathcal{I}^2$, at the end of this round, there is a probability of $\frac{s}{d+1}$ that e_i' belongs to the random sample. Thus, with probability at least $\frac{s}{d+1}$, the sample at the coordinator after observing $\mathcal{I}^1$ is different from the sample after observing $\mathcal{I}^2$.

However, in this round, the coordinator observes a change in execution between $\mathcal{I}^1$ or $\mathcal{I}^2$ only if either (1) site i sends a message to the coordinator in $\mathcal{I}^1$, or (2) site i sends a message to the coordinator in $\mathcal{I}^2$. The probability that at least one of the above events happen is less than $\frac{s}{d+1}$, using the union bound. Further, the behavior of the other sites has an identical distribution on both inputs, since they did not receive any element.

Thus, we have that the contents of the random sample at the end of the round are different with probability at least $\frac{s}{d+1}$, but the probability that the messages observed by the coordinator are different is less than $\frac{s}{d+1}$. This leads to a non-zero probability that the random sample at the coordinator is incorrect at the end of this round, and hence a contradiction. Hence, there can be at most one element e_i such that upon receiving an element $e \in [m] - e_i - \mathcal{D}$, the probability of sending a message to the coordinator is at least $\frac{s}{2(d+1)}$, and the lemma follows. ■

Lemma 7. *Suppose the set of distinct elements observed so far is $\mathcal{D}$, and let $d = |\mathcal{D}|$, and $d \geq s$. For any algorithm $\mathcal{A}$, there exists another round of input $\mathcal{I}(\mathcal{D})$ such that after observing $\mathcal{I}(\mathcal{D})$, the sites will send at least an expected $\frac{ks}{2(d+1)}$ elements to the coordinator.*

Proof: The input $\mathcal{I}(\mathcal{D})$ is constructed as follows. Let e be any element such that $e \notin \left(\mathcal{D} \cup \left(\cup_{i=1}^{k}\{e_i^{\mathcal{D}}\}\right)\right)$. Element e is given to every site in this round. Using Lemma 6, we get that each site $i = 1 \ldots k$ will send a message to the coordinator with probability at least $\frac{s}{2(d+1)}$. The lemma follows. ■

Theorem 2. *For any algorithm $\mathcal{A}$, there exists an input distributed stream, $\mathcal{I}_{\mathcal{A}}$ with d distinct elements such that the expected number of messages sent by the algorithm upon receiving $\mathcal{I}_{\mathcal{A}}$ is at least $\frac{ks}{2}\left(H_d - H_s + 1\right) \approx \frac{ks}{2}\ln\left(\frac{de}{s}\right)$.*

Proof: Input $\mathcal{I}_{\mathcal{A}}$ is constructed as follows. Let $\mathcal{D}_0 = \emptyset$. The input in the first round is $\mathcal{I}(\mathcal{D}_0)$. For $i \geq 0$, let the set of all distinct elements observed till (and including) round i be $\mathcal{D}_i$. Note that the size of $\mathcal{D}_i$ is exactly i. For $i = 0 \ldots (d-1)$, the input in round $i+1$ is $\mathcal{I}(\mathcal{D}_i)$.

From Lemma 7, in round $i > s$, the expected number of messages sent to the coordinator is at least $\frac{ks}{2(i+1)}$. Summing this over all rounds $s+1,\ldots,d$, we get the number of messages to be at least $\frac{ks(H_d-H_s)}{2}$. For rounds 1 till s, similar methods yield that the expected number of messages sent in each round must be at least $\frac{k}{2}$, for the above input. Thus the expected total number of messages sent by this algorithm is at least $\frac{ks}{2}\left(H_d - H_s + 1\right)$. ∎

B. Sampling With Replacement

Thus far, we have considered distinct sampling without replacement. In sampling with replacement, the s different samples are all chosen independently and randomly from the set of distinct elements observed so far, $\mathcal{D}(t)$. A possible solution to distinct sampling with replacement is to repeat s parallel copies of the single element sampling algorithm, each copy using a different hash function. The correctness of this scheme is trivial, and the message cost is s times the cost of a single element sampling algorithm, which is $O(ks \log de)$.

We can do better, using the following observation. When a node i receives an element e, it sends only a single copy of e to the coordinator if it was sampled by one or more of the s parallel copies (of the single element sampling algorithm). We sketch the analysis of this algorithm. For site i and the jth distinct element received by the site, let Y_i^j be a random variable equal to 1 if the site communicated with the coordinator upon receiving this element, and 0 otherwise. The probability that the element is sampled by any of the s different copies of the algorithm is $1-\left(1-\frac{1}{j}\right)^s$. Hence, $E[Y_i^j] = 1-\left(1-\frac{1}{j}\right)^s$. If Y_i denotes the total number of messages sent by site i during execution, we have using linearity of expectation that $E[Y_i] = \sum_{j=1}^{d_i}\left(1-\left(1-\frac{1}{j}\right)^s\right)$. Simplifying this expression (details can be found in the full version of the paper), we get $E[Y_i] \le s\ln(d_i e/s)$. Thus the expected total number of messages in the system is bounded by $O(ks\log(de/s))$, even for the algorithm for sampling with replacement.

We also note the following reduction from distinct sampling without replacement to distinct sampling with replacement. From a distinct sample with replacement of size slightly greater than s, we select a random sample without replacement. It is possible to prove that the result is indeed a distinct sample without replacement from the original dataset. Thus, the lower bound of $\Omega(ks\log(\frac{d}{s}))$ applies to both sampling with and without replacement, and our method leads to an algorithm for sampling with replacement with optimal message complexity.

IV. Sliding Window

In the sliding windows setting, a sample is desired over only a window of most recent elements of the stream. We assume that time is divided into "slots" that are numbered consecutively in an increasing sequence. It is assumed that time is synchronized across the sites so that when site 1 is observing elements in slot t, other sites are also observing elements in the same slot. Let $w > 0$ be an integer denoting the size of the window. The problem is as follows: *when a query is issued to the coordinator at slot t, it returns a random sample of size s chosen without replacement from the set of all distinct elements observed in slots $(t-w+1)$ till t, both endpoints inclusive.* We use the terms "slot" and "time" interchangeably in the following discussion.

Algorithm Idea: At time t, let $\mathcal{S}_i(t,w)$ denote the elements that have arrived in slots $(t-w+1)$ till t at site i. Let $\mathcal{S}(t,w)$ denote the (multiset) union of $\mathcal{S}_i(t,w), i = 1\ldots k$. Let $\mathcal{D}_i(t,w)$ denote the set of distinct elements in $\mathcal{S}_i(t,w)$, and $\mathcal{D}(t,w)$ the set of distinct elements in $\mathcal{S}(t,w)$. For a set of elements E, let $h(E) = \{h(e) | e \in E\}$. The high-level algorithm idea is to choose the elements with the s smallest hash values from $\mathcal{D}(t,w)$. Let $u(t,w)$ denote the s-th smallest hash value from $h(\mathcal{D}(t,w))$.

Implementation of the Idea: The challenge in implementing the sliding windows scenario, when compared with the infinite window scenario, is that $u(t,w)$ is not monotonically decreasing. As elements expire from the window, the value of $u(t,w)$ may increase, and an algorithm has to keep track of this at both the coordinator and the sites. One possible method is as follows. Each site i, at time t, keeps track of the set of s elements with the smallest hash values, chosen from $\mathcal{S}_i(t,w)$. Note that this also forms a distinct sample from $\mathcal{S}_i(t,w)$. Whenever this sample changes, the coordinator is informed. Since the coordinator has all elements with the s smallest hash values from each site, it can maintain elements with the globally s smallest hash values from $\mathcal{D}(t,w)$, and hence a distinct sample from $\mathcal{S}(t,w)$. The message complexity of this algorithm depends on how often the local samples at the individual sites change.

We can reduce the communication cost of the above algorithm by incorporating feedback from the coordinator to the site as follows. Similar to infinite window, the coordinator maintains a variable u, which has the value of $u(t,w)$ at all times. When the coordinator replies back to a site, it conveys the current value of u. However, note that u may increase at the coordinator, due to elements expiring from the window, and this needs also to be conveyed to the sites (note that this was not necessary for the infinite window case). One way to achieve this is that each time u increases, the coordinator broadcasts the new value of u to all nodes. While this algorithm is correct, a broadcast based method can be expensive, since it communicates with nodes which may not be actively receiving messages. We instead use an alternate approach, where the coordinator replies back with the value of u to a site only when the site communicates with the coordinator.

Each site i also needs to maintain a local sliding window sample, consisting of the s smallest hash values within $\mathcal{D}_i(t,w)$. It is known that in general, maintaining the kth smallest element within a sliding window requires space linear in the window size in the worst case [32]. However, in our case, we do not need to maintain the minima over arbitrary numbers, but need to do so over a sequence of random numbers. Hence, we can use the idea from priority sampling similar to [17] to significantly reduce the space consumption at each site. For elements e, e' and time t, t', we say that tuple (e, t) dominates tuple (e', t') at site i if $t > t'$ and $h(e) < h(e')$. Let $\tau_i(e)$ denote the most recent time when e was observed at site i. For elements e, e', we say e dominates e' at site i if $(e, \tau_i(e))$ dominates $(e', \tau_i(e'))$. Each site i has a data structure T_i consisting of all elements that could potentially be included within the random sample of distinct elements either now, or in the future. T_i can be maintained efficiently using a Treap [33].

We omit detailed descriptions of the algorithms due to space constraints, and present the main results on the performance of the algorithms.

Theorem 3. *There is a distributed algorithm that continuously maintains a distinct random sample of size s over a time-based sliding window of size w, with k sites and T total time steps. Suppose $M = \min_{i,t}\{|\mathcal{D}_i(t, w)|\}$. The expected number of message transmissions between the sites and the coordinator is bounded by $2ksT/M$.*

We also prove the following lower bound on the message complexity of *any* algorithm for continuous distinct sampling over a sliding window. In particular, we show that the linear dependence of the message complexity on $s \cdot T$ is necessary.

Lemma 8. *Suppose there is only one site and a coordinator. There exists an input stream $\mathcal{S}$ such that for any distributed algorithm $\mathcal{A}$ that can continuously maintain a distinct random sample at the coordinator over T steps, the expected number of messages sent to the coordinator is at least $\Omega\left(\frac{sT}{w} + s \log \frac{w}{s}\right)$.*

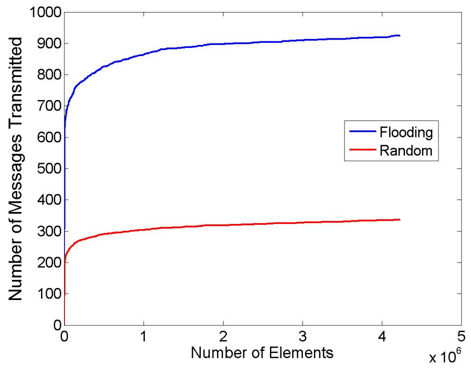
Lemma 9. *The expected space complexity of the sliding window algorithm at site i at time t is $O(s \log |\mathcal{D}_i(t, w)|)$ words.*

V. Experiments

We present an experimental evaluation of our proposed algorithms. We used two datasets. The first is an OC48 Internet Traces Dataset [34], which has anonymous traffic traces taken at a US west coast OC48 peering link for a large ISP in 2002 and 2003. To generate an element, we consider the concatenation of the sender's IP address and the receiver's IP address. The other is the Enron Email Dataset [35], where an element is constructed by concatenating the sender's email address and the receiver's email address. A summary of the data is shown in Table I. Each data point presented is the average of 20 independent runs. We implemented the algorithms in Java, using the MurmurHash [36] hash function. Due to space constraints, we only present the experimental results for the OC48 dataset. The results for the Enron dataset can be found in the full version of the paper [30].

	# Elements	# Distinct
OC48	42,268,510	4,337,768
Enron	1,557,491	374,330

Table I
THE NUMBER OF ELEMENTS AND DISTINCT ELEMENTS IN OC48 IP AND ENRON E-MAIL DATASETS



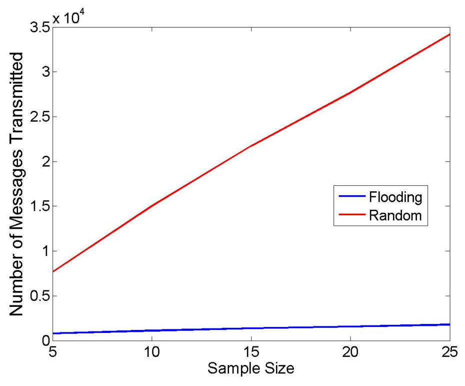
(a) OC48 IP Dataset

Figure 1. The number of messages as a function of number of elements under two different methods of data distribution, "flooding" and "random", for 10 sites and a sample size of 5.

Our theoretical analysis was for the worst case, when the input data can be distributed in an arbitrary manner by the adversary. Here we present experimental results for the infinite windows case, including the impact of data distribution, of sample size, and a comparison with an alternate, natural algorithm that is based on a broadcast from the coordinator to the sites. Due to space constraints, we do not present the results of the sliding windows experiments here. They can be found in the full version.

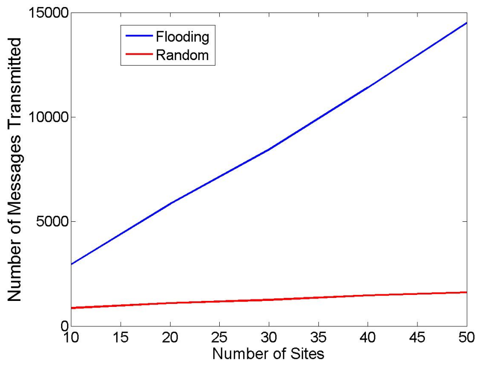
Impact of Data Distribution: We examine the performance of our algorithm under different distribution methods. In the first method, called "flooding", each incoming element is assigned to every site. In the second method, called "random", an incoming element is sent to a single site, chosen uniformly at random. In the third method, "round-robin", each element is sent to a single site, and the elements are assigned to sites in a round-robin manner, i.e. the j-th element is monitored by site $(j \mod k) + 1$. The results for random and round-robin turned out to be nearly identical, so we only present the results for random distribution.

Figure 1 shows the number of messages as the function of the number of elements for 10 sites and a sample size of 5. From Figure 1, we observe that at the beginning, the number of messages increases quickly, since the sample is changing often. As more elements are observed, the rate of change of the sample decreases and fewer messages are sent. It is clear



(a) OC48 IP Dataset

Figure 2. The number of messages as a function of the sample size s for 50 sites.



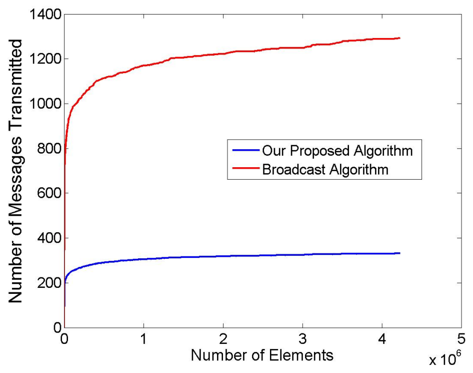
(a) OC48 IP Dataset

Figure 3. The number of messages as function of the number of sites k for a sample size of 20.

that the number of messages under flooding is significantly larger than the number of messages under random, though the total number of distinct elements seen in both inputs is the same. This scenario is explained by our tighter upper bound $2ks\left(1+\sum_{i=1}^{k}\ln\frac{d_i}{s}\right)$ (see Observation 1), which shows the influence of the number of distinct elements observed at the individual sites on the total messages sent.

Impact of Sample Size: Figure 2 shows the number of messages as a function of the sample size for 50 sites; the message complexity increases almost linearly with the sample size, though the slopes are different for different methods of data distribution. Figure 3 shows the number of messages as a function of the number of sites k for a sample size of 20. For flooding, the number of messages increases linearly with the number of sites. However, for random distribution, the number of messages is much smaller than in case of flooding, and is almost independent of the number of sites.

Comparison with Other Algorithms: To our knowledge, there are no prior published methods for distinct sampling on a distributed stream. We compare the performance of our algorithm with another natural algorithm, which we call Algorithm "Broadcast". The difference between Algorithm Broadcast and our proposed method is that Algorithm Broadcast will broadcast the current value of u (which has the value of $u(t)$ at time t) to all sites whenever there is an update to u. This version has the advantage that fewer messages are sent from the sites to the coordinator, since the u_is are always in sync with the coordinator. However, this has the downside of requiring a broadcast each time u changes. In Figure 4, we present a comparison between the two algorithms for 10 sites and a sample size of 5. It is clear that Broadcast requires significantly more messages than our algorithm; this suggests that typically it is not worth keeping the different sites synchronized with respect to the value of u. A "lazy" approach of refreshing u when necessary results in fewer messages.

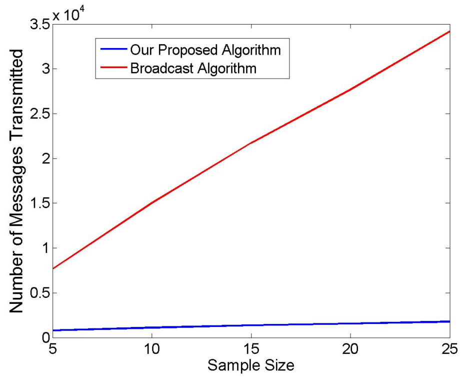


(a) OC48 IP Dataset

Figure 4. The number of messages sent by Algorithm Broadcast and our proposed method for 10 sites and a sample size of 5.

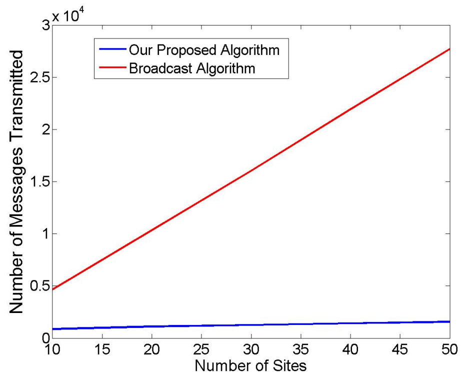
We also note that the message cost of Algorithm Broadcast is linear in the number of sites k, and the sample size s. However, the slope of the Broadcast algorithm is considerably higher. We show the comparison of the two algorithms as a function of the sample size in Figure 6. Similar results are observed with the dependence on the number of sites.

Impact of Skew: We next consider the influence of the non-uniformity in the sizes of the streams observed at different sites. Here, we construct a distributed input which is skewed towards a single site that "dominates" over the other sites in terms of the number of distinct elements that it observes. Each input element is sent to only one site; but instead of dealing it randomly or in a round-robin fashion, we send the element to site 1 with a probability that is a factor α times the probability that a site other than 1 is chosen. We call this factor as the "skew". For example, if the skew is 200, then site 1 is 200 times more likely to receive an element than another site. Figure 7 shows the relation



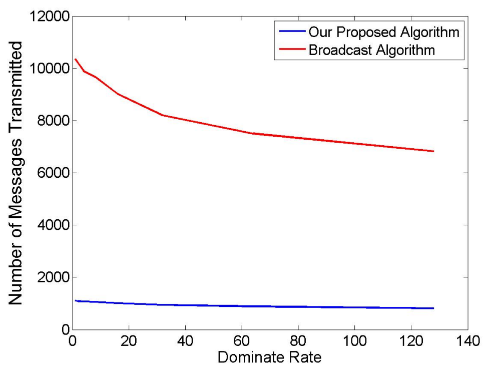
(a) OC48 IP Dataset

Figure 5. The number of messages sent by Algorithm Broadcast and our proposed method on different sample size with 50 sites.



(a) OC48 IP Dataset

Figure 6. The number of messages sent by Algorithm Broadcast and our proposed method on different number of sites (k) with a sample size of 20.



(a) OC48 IP Dataset

Figure 7. The number of messages sent by Algorithm Broadcast and our proposed method, as a function of the skew, for 20 sites and a sample size of 20.

between number of message transmissions and the skew for different algorithms with 20 sites and a sample size of 20. The number of messages transmitted reduces as the skew increases; this can be anticipated through Observation 1 that presents an analysis in terms of the number of distinct elements observed at each site. Note that the higher the skew, the closer this gets to centralized stream monitoring.

VI. Conclusion

We present new message-efficient algorithms for distinct random sampling on a distributed data stream. Our algorithms are practical, easy to implement, and have near-optimal expected message complexity. Surprisingly, the expected message complexity of maintaining a distinct random sample from a distributed manner is inherently greater than that of maintaining a simple random sample from a distributed stream.

Our analysis of the algorithms has used the synchronous model of a distributed system, since it is clearer to analyze the message complexity of a distributed computation within a synchronous system than within an asynchronous system. However, our algorithms are not dependent on there being synchronous lock-step execution of processors within the system. In particular, the paradigm of choosing the element with the smallest hash value as the random sample continues to work in the presence of timing inconsistencies, even if messages are delayed or reordered during transmission.

Some directions for future work are as follows. Our lower bounds for sliding windows are not tight, and it is interesting to obtain an optimal algorithm for DDS on sliding windows either through a better lower bound or a better algorithm. It is also interesting to derive distributed sampling algorithms for more complex structures on data, such as structures on graphs.

References

[1] P. B. Gibbons, "Distinct sampling for highly-accurate answers to distinct values queries and event reports," in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, 2001, pp. 541–550.

[2] G. Cormode and M. Garofalakis, "Sketching streams through the net: distributed approximate query tracking," in *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB Endowment, 2005, pp. 13–24.

[3] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proceedings of the Network and Distributed System Security Symposium, NDSS*, 2005.

[4] A. Poddar, "One pass distinct sampling," http://jonathanlewis.files.wordpress.com/2011/12/one-pass-distinct-sampling.pdf.

[5] T. Johnson, S. Muthukrishnan, and I. Rozenbaum, "Sampling algorithms in a stream operator," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005, pp. 1–12.

[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "Blinkdb: queries with bounded errors and bounded response times on very large data," in *Eighth Eurosys Conference*, 2013, pp. 29–42.

[7] IBM Corporation, "Infosphere streams," http://www-03.ibm.com/software/products/en/infosphere-streams, accessed Jan 2014.

[8] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 423–438.

[9] Apache Software Foundation, "Storm: Distributed and fault-tolerant realtime computation," http://incubator.apache.org/projects/storm.html.

[10] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Continuous sampling from distributed streams," *Journal of the ACM*, vol. 59, no. 2, pp. 10:1–10:25.

[11] D. P. Woodruff and Q. Zhang, "Tight bounds for distributed functional monitoring," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12, pp. 941–960.

[12] S. Tirthapura and D. P. Woodruff, "Optimal random sampling from distributed streams revisited," in *Proc. International Symposium on Distributed Computing (DISC)*, 2011, pp. 283–297.

[13] G. Cormode, "The continuous distributed monitoring model," *SIGMOD Record*, vol. 42, no. 1, pp. 5–14, May 2013.

[14] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[15] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Information Processing Letters*, vol. 97, no. 5, pp. 181 – 185, 2006.

[16] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," *Journal of Computer and System Sciences*, vol. 78, pp. 260 – 272, 2012.

[17] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 633–634.

[18] R. Gemulla and W. Lehner, "Sampling time-based sliding windows in bounded space," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, pp. 379–392.

[19] C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proceedings of the 32nd International Conference on Very Large Databases*, 2006, pp. 607–618.

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

[21] S. Ganguly, M. Garofalakis, and R. Rastogi, "Tracking set-expression cardinalities over continuous update streams," *The VLDB Journal*, vol. 13, no. 4, pp. 354–369, 2004.

[22] G. Frahling, P. Indyk, and C. Sohler, "Sampling in dynamic data streams and applications," in *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, ser. SCG '05. ACM, pp. 142–149.

[23] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2008, pp. 1076–1085.

[24] C. Arackaparambil, J. Brody, and A. Chakrabarti, "Functional monitoring without monotonicity," in *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, 2009, pp. 95–106.

[25] K. Yi and Q. Zhang, "Optimal tracking of distributed heavy hitters and quantiles," *Algorithmica*, vol. 65, no. 1, pp. 206–223, 2013.

[26] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *ACM Transactions on Database Systems*, vol. 32, no. 4, 2007.

[27] N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, I. Sharfman, and A. Schuster, "Prediction-based geometric monitoring over distributed data streams," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, pp. 265–276.

[28] B. Lahiri and S. Tirthapura, "Stream sampling," in *Encyclopedia of Database Systems*, 2009, pp. 2838–2842.

[29] P. B. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," in *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '02, pp. 63–72.

[30] Y.-Y. Chung and S. Tirthapura, "Distinct random sampling from a distributed stream," http://www.ece.iastate.edu/~snt/pubs/dds.pdf, Oct 2014.

[31] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," in *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2010, pp. 77–86.

[32] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[33] R. Seidel and C. R. Aragon, "Randomized search trees," in *Algorithmica*, vol. 16, no. 4-5. Springer-Verlag, 1996, pp. 464–497.

[34] CAIDA OC48 Trace Project, "The caida ucsd oc48 internet traces dataset - (2002-2003)," http://imdc.datcat.org/contact/1-0037-M=CAIDA+OC48+Trace+Project, 2006.

[35] CALO Project, "Enron email dataset," http://www.cs.cmu.edu/~enron/.

[36] V. Holub, "Murmur hash 2.0," http://d3s.mff.cuni.cz/~holub/sw/javamurmurhash/MurmurHash.java.