Algorithmica (2019) 81:2222–2243
https://doi.org/10.1007/s00453-018-00531-y

Randomized Algorithms for Tracking Distributed Count, Frequencies, and Ranks

Zengfeng Huang[1] · Ke Yi[2] · Qin Zhang[3]

Received: 27 October 2016 / Accepted: 23 November 2018 / Published online: 11 December 2018

Abstract
We show that randomization can lead to significant improvements for a few fundamental problems in distributed tracking. Our basis is the *count-tracking* problem, where there are k players, each holding a counter n_i that gets incremented over time, and the goal is to track an ε-approximation of their sum $n = \sum_i n_i$ continuously at all times, using minimum communication. While the deterministic communication complexity of the problem is $\Theta(k/\varepsilon \cdot \log N)$, where N is the final value of n when the tracking finishes, we show that with randomization, the communication cost can be reduced to $\Theta(\sqrt{k}/\varepsilon \cdot \log N)$. Our algorithm is simple and uses only $O(1)$ space at each player, while the lower bound holds even assuming each player has infinite computing power. Then, we extend our techniques to two related distributed tracking problems: *frequency-tracking* and *rank-tracking*, and obtain similar improvements over previous deterministic algorithms. Both problems are of central importance in large data monitoring and analysis, and have been extensively studied in the literature.

Keywords Continuous distributed tracking · Randomized algorithms · Distributed streaming

A preliminary version of this article was presented at the ACM Symposium on Principles of Database Systems (PODS), 2012.

✉ Zengfeng Huang
huangzf@fudan.edu.cn

Ke Yi
yike@cse.ust.hk

Qin Zhang
qzhangcs@indiana.edu

1 School of Data Science, Fudan University, Shanghai, China

2 The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

3 Indiana University Bloomington, Bloomington, USA

1 Introduction

We start with a very basic problem in distributed tracking, what we call *count-tracking*. There are k players each holding a counter n_i that is initially 0. Over time, the counters get incremented and we denote by $n_i(t)$ the value of the counter n_i at time t. The goal is to track an ε-approximation of the total count $n(t) = \sum_i n_i(t)$, i.e., an $\hat{n}(t)$ such that $(1-\varepsilon)n(t) \leq \hat{n}(t) \leq (1+\varepsilon)n(t)$,[1] continuously at all times. There is a coordinator whose job is to maintain such an $\hat{n}(t)$, and will try to do so using minimum communication with the k players (the formal model of computation will be defined shortly).

There is a trivial solution to the count-tracking problem: Every time a counter n_i has increased by a $1+\varepsilon$ factor, the player informs the coordinator of the change. Thus, the coordinator always has an ε-approximation of every n_i, hence an ε-approximation of their sum n. Letting N denote the final value of n, simple analysis shows that the communication cost of this algorithm is $O(k/\varepsilon \cdot \log N)$.[2] This algorithm was actually used in [16] for solving essentially the same problem, which also provided many practical motivations for studying this problem. Note that this algorithm is deterministic and only uses one-way communication (from the players to the coordinator), and yet it turns out this simple algorithm is already optimal for deterministic algorithms, even if two-way communication is allowed [29]. Thus the immediate questions are: What about randomized algorithms that are allowed to fail with a small probability? Is two-way communication not useful at all? In this paper, we set out to address these questions, and then move on to consider other related distributed tracking problems.

1.1 The Distributed Tracking Model

We first give a more formal definition of the computation model that we will work with, which is essentially the same as those used in prior work on distributed tracking [2,3,5,7,9,10,16,29]. There are k distributed *sites* $S_1, \ldots, S_k$, each receiving a stream of elements over time, possibly at varying rates. Let N be the total number of elements in all k streams. We denote by $A_i(t)$ the multiset (bag) of elements received by S_i up until time t, and let $A(t) = \biguplus_{i=1}^{k} A_i(t)$ be the combined data set, where $\uplus$ denotes multiset addition. There is a coordinator whose job is to maintain (an approximation of) $f(A(t))$ continuously at all times, for a given function f (e.g., $f(A(t)) = |A(t)|$ for the count-tracking problem above). The coordinator has a direct two-way communication channel with each of the sites; note that broadcasting a message costs k times the communication for a single message. The sites do not communicate with each other directly, but this is not a limitation since they can always pass messages via the coordinator. We assume that communication is instant, i.e., no element will arrive until all parties have decided not to send more messages. As in prior work, our measures of complexity will be the communication cost and the space used to process each stream. Unless otherwise specified, the unit of both measures is a *word*, and we

[1] We sometimes omit "(t)" when the context is clear.

[2] A more careful analysis leads to a slightly better bound of $O(k/\varepsilon \cdot \log(\varepsilon N/k))$, but we will assume that N is sufficiently large, compared to k and $1/\varepsilon$, to simplify the bounds.

assume that any integer less than N, as well as an element from the stream, can fit in one word.

This model was initially abstracted from many applied settings, ranging from distributed data monitoring, wireless sensor networks, to network traffic analysis, and has been extensively studied in the database community. See the recent survey of [6] for more information. From 2008 [9], the model has started to attract interests from the theory community as well, as it naturally combines two well-studied models: the data stream model and multi-party communication complexity. When there is only $k = 1$ site who also plays the role of the coordinator, the model degenerates to the standard streaming model; when $k \geq 2$ and our goal is to do a one-shot computation of $f(A(\infty))$, then the model degenerates to the (number-in-hand) k-party communication model. Thus, distributed tracking is more general than both models. Meanwhile, it also appears to be significantly different from either, with the above count-tracking problem being the best example. This problem is trivial in both the streaming and the communication model (even computing the exact count is trivial), whereas it becomes nontrivial in the distributed tracking model and requires new techniques, especially when randomization is allowed, as illustrated by our results in this paper.

Note that there is some work on *distributed streaming* (see e.g. [12,17]) that adopts a model very similar to ours, but with a fundamental difference. In their model there are k streams, each of which runs a streaming algorithm on its local data. But the function f on the combined streams is computed only at the end or upon requests by the user. As one can see that the count-tracking problem is also trivial in this model. The crucial difference is that, in this model, the sites wait passively to get polled. If we want to track f continuously, we have to poll the sites all the time. Whereas in our model, the sites actively participate in the tracking protocol to make sure that f is always up-to-date.

1.2 Problem Statements, Previous and New Results

In this paper, we first study the count-tracking problem. Then we extend our approach to two related, more general problems: *frequency-tracking* and *rank-tracking*. Both problems are of central importance in large data monitoring and analysis, and have been extensively studied in the literature. In all the communication upper bounds, we will assume $k \leq 1/\varepsilon^2$; otherwise all of them will carry an extra additive $O(k \log N)$ term. Our results are summarized in Table 1; below we discuss each of them separately.

As mentioned earlier, the deterministic communication complexity for the count-tracking problem has been settled at $\Theta(k/\varepsilon \cdot \log N)$ [29],[3] with or without two-way communication. In this paper, we show that with randomization *and* two-way communication, this is reduced to $\Theta(\sqrt{k}/\varepsilon \cdot \log N)$. We first in Sect. 2.1 present a randomized algorithm with this communication cost that, at *any one* given time instance, maintains an ε-approximation of the current n with a constant probability. The algorithm is very simple and uses $O(1)$ space at each site. It is easy to make the algorithm correct for *all* time instances and boost the probability to $1 - \delta$: Since we can use the same

[3] The lower bound in [29] was stated for the heavy hitters tracking problem, but essentially the same proof works for count-tracking.

Table 1 Space and communication costs of previous and new algorithms

		Space (per site)	Communication
Count-tracking	Trivial	$O(1)$	$\Theta(k/\varepsilon \cdot \log N)$
	New	$O(1)$	$O(\sqrt{k}/\varepsilon \cdot \log N)$
			$\Omega(\sqrt{k}/\varepsilon \cdot \log N)$ messages
Frequency-tracking	[29]	$O(1/\varepsilon)$	$\Theta(k/\varepsilon \cdot \log N)$
	New	$O(1/(\varepsilon\sqrt{k}))$	$O(\sqrt{k}/\varepsilon \cdot \log N)$
		$\Omega(1/(\varepsilon\sqrt{k}))$ bits*	$\Omega(\sqrt{k}/\varepsilon \cdot \log N)$ messages
Rank-tracking	[29]	$O(1/\varepsilon \cdot \log n)$	$O(k/\varepsilon \cdot \log N \log^2(1/\varepsilon))$
	New	$O\left(\frac{1}{\varepsilon\sqrt{k}} \log^{1.5} \frac{1}{\varepsilon} \log^{0.5} \frac{1}{\varepsilon\sqrt{k}}\right)$	$O\left(\sqrt{k}/\varepsilon \cdot \log N \log^{1.5} \frac{1}{\varepsilon\sqrt{k}}\right)$
		$\Omega(1/(\varepsilon\sqrt{k}))$ bits*	$\Omega(\sqrt{k}/\varepsilon \cdot \log N)$ messages
Sampling	[10]	$O(1)$	$O(1/\varepsilon^2 \cdot \log N)$

We assume $k \le 1/\varepsilon^2$. All upper bounds are in terms of words
*This is conditioned upon the communication cost being $O(\sqrt{k}/\varepsilon \cdot \log N)$ bits

approximate value $\hat{n}$ of n until n grows by a $1+\varepsilon$ factor, it suffices to make the algorithm correct for $O(\log_{1+\varepsilon} N) = O(1/\varepsilon \cdot \log N)$ time instances. Then running $O(\log(\frac{\log N}{\delta\varepsilon}))$ independent copies of the algorithm and taking the median will achieve the goal of tracking n continuously at all times, with probability at least $1-\delta$. The $\Omega(\sqrt{k}/\varepsilon \cdot \log N)$ lower bound (Sect. 2.2) actually holds on the number of messages that have to be exchanged, regardless of the message size, and holds even assuming the sites have unlimited space and computing power. That randomization is necessary to achieve this $\sqrt{k}$-factor improvement follows from the previous deterministic lower bound [29]; here in Sect. 2.2 we give a proof that two-way communication is also required. More precisely, we show that any randomized algorithm with one-way communication has to use $\Omega(k/\varepsilon \cdot \log N)$ communication, i.e., the same as that for deterministic algorithms.

In the *frequency-tracking* (a.k.a. *heavy hitters tracking*) problem, $A(t)$ is a multiset of cardinality $n(t)$ at time t. Let $f_j(t)$ be the frequency of element j in $A(t)$. The goal is to maintain a data structure from which $f_j(t)$, for any given j, can be estimated with absolute error at most $\varepsilon n(t)$, with probability at least 0.9 (say). Note that this problem degenerates to count-tracking when there is only one element. It is reasonable to ask for an error in terms of $n(t)$: if the error were $\varepsilon f_j(t)$, then every element would have to be reported if they were all distinct. In fact, this error requirement is the widely accepted definition for the heavy hitters problem, which has been extensively studied in the streaming literature [8]. Several algorithms with the optimal $O(1/\varepsilon)$ space exist [18–20]. In the distributed tracking model, we previously [29] gave a deterministic algorithm with $O(k/\varepsilon \cdot \log N)$ communication, which is the best possible for deterministic algorithms. In this paper, by generalizing our count-tracking algorithm, we reduce the cost to $O(\sqrt{k}/\varepsilon \cdot \log N)$, with randomization (Sect. 3). Since this problem is more general than count-tracking, by the count-tracking lower bound, this is also optimal. Our algorithm uses $O(1/(\varepsilon\sqrt{k}))$ space to process the stream at each site, which is actually smaller than the $\Omega(1/\varepsilon)$ space lower bound for this problem in the streaming model. This should not come as a surprise: Due to the fact that the site is allowed to communicate to the coordinator *during* the streaming process, the streaming lower bounds do not apply in our model. To this end, we prove a new space lower bound of $\Omega(1/(\varepsilon\sqrt{k}))$ bits for our model, showing that our algorithm also uses near-optimal space. This space lower bound is conditioned upon the requirement that the communication cost should be $O(\sqrt{k}/\varepsilon \cdot \log N)$ bits. Note that it is not possible to prove a space lower bound unconditional of communication: A site can send every element to the coordinator and thus only needs $O(1)$ space. In fact, what we prove is a space-communication trade-off; please see Sect. 3.2 for the precise statement.

For the *rank-tracking* problem, it will be convenient to assume that the elements are drawn from a totally ordered universe and $A(t)$ contains no duplicates. The *rank* of an element x in $A(t)$ (x need not be in $A(t)$) is the number of elements in $A(t)$ smaller than x, and our goal is to compute a data structure from which the rank of any given x can be estimated with error at most $\varepsilon n(t)$, with constant probability. Note that a rank-tracking algorithm also solves the frequency-tracking problem (but not vice versa), by turning each element x into a pair (x, y) to break all ties (by comparing the pairs lexicographically) and maintaining such a rank-tracking data structure. When the frequency of x is desired, we ask for the ranks of $(x, 0)$ and (x, ∞) and take the difference. We previously [29] gave a deterministic algorithm for the rank-tracking problem with communication $O(k/\varepsilon \cdot \log N \log^2(1/\varepsilon))$. In this paper, we show in Sect. 4 how randomization can bring this down to $O(\sqrt{k}/\varepsilon \cdot \log N \log^{1.5}(1/\varepsilon\sqrt{k}))$, which is again optimal ignoring polylog$(1/\varepsilon, k)$ factors. Since rank-tracking is more general than frequency-tracking, the previous lower bounds also hold here. Our algorithm uses space that is also close to the $\Omega(1/(\varepsilon\sqrt{k}))$ lower bound.

Since we are talking about randomized algorithms with a constant success probability, we should also compare with random sampling. It is well known [25] that this probabilistic guarantee can be achieved for all the problems above by taking a random sample of size $O(1/\varepsilon^2)$. A random sample can be maintained continuously over distributed streams [10,24], solving these distributed tracking problems, with a communication cost of $O(1/\varepsilon^2 \cdot \log N)$. This is worse than our algorithms when $k = o(1/\varepsilon^2)$. As noted earlier, all the upper bounds we have mentioned above have a hidden additive $O(k \log N)$ term. Thus when $k = \Omega(1/\varepsilon^2)$, all of them boil down to $O(k \log N)$,[4] so our results are more interesting for the $k \le 1/\varepsilon^2$ case, which we will assume in all the upper bounds throughout the paper. In the lower bound statements, however, we do distinguish between the two cases. The lower bounds in Table 1 assume $k \le 1/\varepsilon^2$, and they all match the upper bounds (except for the rank-tracking problem); for the $k = \Omega(1/\varepsilon^2)$ case, the lower bound is $\Omega(k)$ (Theorem 3), which leaves a gap of $\Theta(\log N)$ from the upper bound.

The idea behind all our algorithms is very simple. Instead of deterministic algorithms, we use randomized algorithms that produce unbiased estimators for n_i, the frequencies, and ranks with variance $(\varepsilon n)^2/k$, leading to an overall variance of $(\varepsilon n)^2$, which is sufficient to produce an estimate within error εn with constant probability. This means we can afford an error of $\varepsilon n/\sqrt{k}$ from each site, as opposed to $\varepsilon n/k$ for

[4] The bound of the random sampling algorithm [10,24] is actually slightly better, which is $O(k \log N/\log(k\varepsilon^2))$.

deterministic algorithms. This is essentially where we obtain the $\sqrt{k}$-factor improvement by randomization. Our algorithms are simple and extremely lightweight, in particular the count-tracking and frequency-tracking algorithms, thus can be easily implemented in power-limited distributed systems like wireless sensor networks.

1.3 Other Related Work

As distributed tracking is closely related to the streaming and the k-party communication model, it could be enlightening to compare with the known results of the above problems in these models. As mentioned earlier, the count-tracking problem is trivial in both models, requiring $O(1)$ space in the streaming model and $O(k)$ communication in the k-party communication model.

Both the frequency-tracking and rank-tracking problems have been extensively studied in the streaming model with a long history. The former was first resolved by the MG algorithm [20] with the optimal space $O(1/\varepsilon)$, though several other algorithms with the same space bound have been proposed later on [18,19]. The rank problem is also one of the earliest problems studied in the streaming model [21]. The best deterministic algorithm to date is the one by Greenwald and Khanna [13]. It uses $O(1/\varepsilon \cdot \log n)$ working space to maintain a structure of size $O(1/\varepsilon)$, from which any rank can be estimated with error εn. Note that the rank problem is often studied as the *quantiles* problem in the literature. Recall that for any $0 \le \phi \le 1$, the ϕ-quantile of D is the element in $A(t)$ that ranks at $\lfloor \phi n \rfloor$, while an ε-approximate ϕ-quantile is any element that ranks between $(\phi - \varepsilon)n$ and $(\phi + \varepsilon)n$. Clearly, if we have the data structure for one problem, we can do a binary search to solve the other. Thus the two problems are equivalent, for deterministic algorithms. For algorithms with probabilistic guarantees, we need all $O(\log(1/\varepsilon))$ decisions in the binary search to succeed, which requires the failure probability to be lowered by an $O(\log(1/\varepsilon))$ factor. By running $O(\log\log(1/\varepsilon))$ independent copies of the algorithm, this is not a problem. So the two problems differ by at most a factor of $O(\log\log(1/\varepsilon))$.

The existing streaming algorithms for the frequency and rank problems can be used to solve the one-shot version of the problem in the k-party communication model easily. More precisely, we use a streaming algorithm to summarize the data set at each site with a structure of size $O(1/\varepsilon)$, and then send these summary structures to the coordinator, resulting in a communication cost of $O(k/\varepsilon)$. Recently, we designed randomized algorithms for these two problems with $O(\sqrt{k}/\varepsilon)$ communication [14,15], which have just been shown to be near-optimal [27]. Here we have extended the one-shot algorithms of [14,15] to the continuous tracking setting. The results have demonstrated that, the seemingly more challenging tracking problem, which requires us to solve the one-shot problem continuously at all times, is only harder by an $\Theta(\log N)$ factor than the one-shot version (except for the count-tracking problem, which is much harder than its one-shot version).

Finally, we should mention that all these distributed tracking problems have been studied in the database community previously, but mostly using heuristics. Keralapura et al. [16] approached the count-tracking problem using prediction models, which do not work under adversarial inputs. Babcock and Olston [3] studied the top-k tracking problem, a variant of the frequency (heavy hitters) tracking problem, but did not offer a theoretical analysis. The rank-tracking problem was first studied by Cormode et al. [7]; their algorithm has a communication cost of $O(k/\varepsilon^2 \cdot \log N)$ under certain inputs.

2 Tracking Distributed Count

2.1 The Algorithm

The algorithm with a fixed p Let p be a parameter to be determined later. For now we will assume that p is fixed. The algorithm is very simple: Whenever site S_i receives an element (hence n_i gets incremented by one), it sends the latest value of n_i to the coordinator with probability p. Let $\bar{n}_i$ be the last updated value of n_i received by the coordinator. We first estimate each n_i by

$$\hat{n}_i = \begin{cases} \bar{n}_i - 1 + 1/p, & \text{if } \bar{n}_i \text{ exists;} \\ 0, & \text{else.} \end{cases} \tag{1}$$

Then we estimate n as $\hat{n} = \sum_i \hat{n}_i$.
Analysis As mentioned in the introduction, our analysis will hold for any given one time instance. It is also important to note that this given time instance shall not depend on the randomization internal to the algorithm.

We show that each $\hat{n}_i$ is an unbiased estimator of n_i with variance at most $1/p^2$. This is very intuitive, since $n_i - \bar{n}_i$ is the number of failed trials until the site decides to send an update to the coordinator, when we look backward from the current time instance. This follows a geometric distribution with parameter p, but not quite, as it is bounded by n_i. This is why we need to separate the two cases in (1). The following calculation appeared in [14]; we include it here for completeness.

Lemma 1 *$E[\hat{n}_i] = n_i$; $Var[\hat{n}_i] \le 1/p^2$.*

Proof Define the random variable

$$X = \begin{cases} n_i - \bar{n}_i + 1, & \text{if } \bar{n}_i \text{ exists;} \\ n_i + 1/p, & \text{else.} \end{cases}$$

Now we can rewrite $\hat{n}_i$ as $\hat{n}_i = n_i - X + 1/p$. Thus it suffices to show that $\mathsf{E}[X] = 1/p$ and $\mathsf{Var}[X] \le 1/p^2$. Letting $t = n_i - \bar{n}_i + 1$, we have

$$\mathsf{E}[X] = \sum_{t=1}^{n_i} (t(1-p)^{t-1}p) + (n_i + 1/p)(1-p)^{n_i} = \frac{1}{p}.$$

$$\begin{aligned}\mathsf{Var}[X] &= \sum_{t=1}^{n_i} ((t - 1/p)^2(1-p)^{t-1}p) + (n_i + 1/p - 1/p)^2(1-p)^{n_i} \\ &= \frac{(1-p)(1-(1-p)^{n_i})}{p^2} \le \frac{1}{p^2}.\end{aligned}$$

□

By Lemma 1, we know that $\hat{n}$ is an unbiased estimator of n with variance $\le k/p^2$. Thus, if $p = \sqrt{k}/\varepsilon n$, the variance of $\hat{n}$ will be $(\varepsilon n)^2$, which means that $\hat{n}$ has error at most $2\varepsilon n$ with probability at least 3/4, by Chebyshev inequality. Rescaling ε and p by a constant will reduce the error to εn and improves the success probability to 0.9, as desired.

Previously, we used similar ideas to solve the *one-shot* quantile problem over distributed data [14]. Here, we essentially treat the numerical values of the items in the quantile problem as the timestamps in the count-tracking problem, and simulate the one-shot sampling algorithm of [14] in the continuous setting.

Dealing with a decreasing p. It is not possible and necessary to set p exactly to $\sqrt{k}/\varepsilon n$. From the analysis above, it should be clear that keeping $p = \Theta(\sqrt{k}/\varepsilon n)$ will suffice. To do so, we first track n within a constant factor. This can be done efficiently as follows. Each site S_i keeps track of its own counter n_i. Whenever n_i doubles, it sends an update to the coordinator. The coordinator sets $n' = \sum_{i=1}^{k} n'_i$, where n'_i is the last update of n_i. When n' doubles (more precisely, when n' changes by a factor between 2 and 4), the coordinator broadcasts n' to all the sites. Let $\bar{n}$ be the last broadcast value of n'. It is clear that $\bar{n}$ is always a constant-factor approximation of n. The communication cost is $O(k \log N)$, since each site sends $O(\log N)$ updates to the coordinator and the coordinator broadcasts $O(\log N)$ times, each of which costs k messages. These broadcasts divide the whole tracking period into $O(\log N)$ rounds, and within each round, n stays within a constant factor of $\bar{n}$, the broadcast value at the beginning of the round.

Now, when $\bar{n} \le \sqrt{k}/\varepsilon$, all the sites set $p = 1$. This causes all the first $O(\sqrt{k}/\varepsilon)$ elements to be sent to the coordinator. When $\bar{n} > \sqrt{k}/\varepsilon$, each site sets $p = \frac{1}{\lfloor \varepsilon \bar{n}/\sqrt{k} \rfloor_2}$, where $\lfloor x \rfloor_2$ denotes the largest power of 2 smaller than x. Since $\bar{n}$ is monotonically increasing, p gets halved over the rounds. At the beginning of a round, each site can sends the current local count to the coordinator and start a new tracking algorithm with the new sampling probability. The correctness of this algorithm is obvious, while the extra communication cost in each round is k.

It is easy to see that the total communication cost in each round is $O(k + pn) = O(k + \sqrt{k}/\varepsilon) = O(\sqrt{k}/\varepsilon)$, thus the total cost is $O(\sqrt{k}/\varepsilon \cdot \log N)$.

Theorem 1 *There is a randomized algorithm for the count-tracking problem that tracks $n = \sum_i n_i$ within error εn with probability at least 0.9. It uses $O(1)$ space at each site and $O(\sqrt{k}/\varepsilon \cdot \log N)$ total communication.*

2.2 The Lower Bound

Before proving the lower bounds, we first state our lower bound model formally, in the context of the count-tracking problem. The N elements arrive at the k sites in an online fashion at arbitrary time instances. We do not allow spontaneous communication. More precisely, it means that a site is allowed to send out a message only if it has just received an element or a message from the coordinator. Likewise, the coordinator is allowed to send out messages only if it has just received messages from one or more sites. When a site S_j is allowed to send out a message, it decides whether it will indeed send a message, and the content of the message if so, based only on its local counter n_j and the message history between S_j and the coordinator, possibly using some random source. We assume that the site does not look at the current clock. We argue that the clock conveys no information since the elements arrive at arbitrary and unpredictable time instances. (If the elements arrive in a predictable fashion, say, one per time step, the problem can be solved without communication al all.) Similarly, when the coordinator is allowed to send out messages, it makes the decision on where and what to send based only on its message history and some random source. We will lower bound the communication cost only by the number of messages, regardless of the message size.

2.2.1 One-Way Communication Lower Bound

In this section we show that two-way communication is necessary to achieve the upper bound in Theorem 1, by proving the following lower bound. Remember that we assume N is sufficiently larger than k and $1/\varepsilon$.

Theorem 2 *If only the sites can send messages to the coordinator but not vice versa, then any randomized algorithm for the count-tracking problem that, at any time, estimates n within error εn with probability at least 0.9 must send $\Omega(k/\varepsilon \cdot \log N)$ messages.*

Proof We first define the hard input distribution μ.

(a) With probability 1/2, all elements arrive at one site that is uniformly picked at random.
(b) Otherwise, the N elements arrive at the k sites in a round-robin fashion, each site receiving N/k elements in the end.

By Yao's Minimax principle [28], we only need to argue that any deterministic algorithm with success probability at least 0.8 under μ has expected cost $\Omega(k/\varepsilon \cdot \log N)$.

Note that when only one-way communication is allowed, a site decides whether to send messages to the coordinator only based on its local counter n_j. Thus the communication pattern can be essentially described as follows. Each site S_j has a series of thresholds $t_j^1, t_j^2, \ldots$ such that when $n_j = t_j^i$, the site sends the i-th message to the coordinator. These thresholds should be fixed at the beginning.

We lower bound the communication cost by rounds. Let W_i be the number of elements that have arrived up until round i. We divide the rounds by setting $W_1 = k/\varepsilon$, and $W_{i+1} = \lceil (1+\varepsilon) W_i \rceil$ for $i \geq 1$. Thus there are $1/\varepsilon \cdot \log(\varepsilon N/k)$ rounds, which is $\Omega(1/\varepsilon \cdot \log N)$ for sufficiently large N.

At the beginning of round $i+1$, suppose that $S_1, S_2, \ldots, S_k$ have already sent $z_1^i, z_2^i, \ldots, z_k^i$ messages to the coordinator, respectively. Let

$$t_{\max}^{i+1} = (1+\varepsilon) \cdot \max\left\{ t_j^{z_j^i} \mid j = 1, 2, \ldots, k \right\}.$$

We first observe that there must be at least $k/2$ sites with their next threshold $t_j^{z_j^i+1} \leq t_{\max}^{i+1}$. Otherwise, suppose there are less than $k/2$ sites with such next thresholds, then with probability at least $1/4$ case (a) happens and the random site S_j chosen to receive all elements has $t_j^{z_j^i+1} > t_{\max}^{i+1} \geq (1+\varepsilon)t_j^{z_j^i}$. Thus, with probability at least $1/4$ the algorithm fails when the $t_{\max}^{i+1}$-th element arrives, contradicting the success guarantee.

On the other hand, with probability $1/2$ case (b) happens. In this case all $t_j^{z_j^i}$ $(j = 1, 2, \ldots, k)$ are no more than W_i/k, since in case (b), elements arrive at all k sites in turn. In the next εW_i elements, each site S_j receives $\varepsilon W_i/k$ elements. If the site S_j has $t_j^{z_j^i+1} \leq t_{\max}^{i+1}$, then it must send a message in this round, since $W_i/k + \varepsilon W_i/k \geq t_{\max}^{i+1} \geq t_j^{z_j^i+1}$, that is, its (z_j^i+1)-th threshold is triggered. As argued, there are $\geq k/2$ sites with $t_j^{z_j^i+1} \leq t_{\max}^{i+1}$, so the communication cost in this round is at least $k/2$.

Summing up all rounds, the total communication is at least $\Omega(k/\varepsilon \cdot \log N)$. □

2.2.2 Two-Way Communication Lower Bound

Below we prove two randomized lower bounds when two-way communication is allowed. The first one justifies the assumption $k \leq 1/\varepsilon^2$, since otherwise, random sampling will be near-optimal.

Theorem 3 *Any randomized algorithm for the count-tracking problem that, at any time, estimates n within error $0.1n$ with probability at least 0.9 must exchange $\Omega(k)$ messages.*

Proof The hard input distribution is the same as that in the proof of Theorem 2. To prove this lower bound we are only interested in the number of sites that communicate with the coordinator at least once. Before any element arrives, we can still assume that each site keeps a triggering threshold. The thresholds of S_j shall remain the same unless it communicates with the coordinator at least once. We argue that there must be at least $k/2$ sites whose triggering threshold is no more than 1, since otherwise if case (a) happens and the randomly chosen site is one with a triggering threshold larger than 1, the algorithm will fail, which would happen with probability at least $1/4$. On the other hand, if case (b) happens, then all the sites with threshold 1 will have to communicate with the coordinator at least once: either their thresholds are triggered by the round-robin arrival of elements, or they receive a message from the coordinator, which can possibly change their threshold. □

Finally, we show that the upper bound in Theorem 1 is tight. We first introduce the following primitive problem.

Definition 1 (1-*bit*) Let s be either $k/2+\sqrt{k}$ or $k/2-\sqrt{k}$, each with probability $1/2$. From the k sites, a subset of s sites picked uniformly at random each have bit 1, while the other $k-s$ sites have bit 0. The goal of the communication problem is for the coordinator to find out the value of s with probability at least 0.8.

We will show the following lower bound for the 1- BIT problem.

Lemma 2 *Any deterministic algorithm that solves 1-bit has distributional communication complexity $\Omega(k)$.*

Lemma 2 immediately implies the following theorem:

Theorem 4 *Any randomized algorithm for the count-tracking problem that, at any time, estimates n within error εn with probability at least 0.9 must exchange $\Omega(\sqrt{k}/\varepsilon \cdot \log N)$ messages, when $k < 1/\varepsilon^2$.*

Proof We will again fix a hard input distribution first and then focus on the distributional communication complexity of deterministic algorithms with success probability at most 0.8. Let $[m] = \{0, 1, \ldots, m-1\}$. The adversarial input consists of $\ell = \log \frac{\varepsilon N}{k} = \Omega(\log N)$ rounds. We further divide each round $i \in [\ell]$ into $r = 1/(2\varepsilon\sqrt{k})$ subrounds.

The input at round $i \in [\ell]$ is constructed as follows, at each subround $j \in [r]$, we first choose s to be $k/2 + \sqrt{k}$ or $k/2 - \sqrt{k}$ with equal probability. Then we choose s sites out of the k sites uniformly at random and send 2^i elements to each of them (the order does not matter).

It is easy to see that at the end of in each subround in round i, the total number of items is no more than $\tau_i = \sqrt{k}/\varepsilon \cdot 2^i$. Thus after $s \cdot 2^i$ elements have arrived in a subround, the algorithm has to correctly identify the value of s with probability at least 0.8, since otherwise with probability at least 0.2 the estimation of the algorithm will deviate from the true value by at least $\sqrt{k} \cdot 2^i > \varepsilon\tau_i$, violating the success guarantee of the algorithm. This is exactly the 1-bit problem defined above. By Lemma 2, the communication cost of each subround is $\Omega(k)$. Summing over all r subrounds and then all ℓ rounds, we have that the total communication is at least $\ell \cdot r \cdot \Omega(k) \geq \Omega(\sqrt{k}/\varepsilon \cdot \log N)$. □

Now we prove Lemma 2.

Proof (of Lemma 2) First of all, observe that whenever the coordinator communicates with a site, the site can send its whole input (i.e., its only bit) to the coordinator. After that, the coordinator knows all the information about that site and does not need to communicate with it further. Therefore all that we need to investigate is the number of sites the coordinator needs to communicate with.

There can be two types of actions in the protocol.

(a) A site initiates a communication with the coordinator based on the bit it has.
(b) The coordinator, based on all the information it has gathered so far, asks some site to send its bit.

Note that if a type (b) communication takes place before a type (a) communication, we can always swap the two, since this only gives the coordinator more information at an earlier stage. Thus we can assume that all the type (a) communications happen before type (b) ones.

In the first phase where all the type (a) communications happen, let x be the number of sites that send bit 0 to the coordinator, and y be the number of sites that send bit 1 to the coordinator. If $\mathsf{E}[x+y] = \Omega(k)$, then we are done. So let us assume that $\mathsf{E}[x+y] =$

$o(k)$. By Markov inequality we have that, with probability at least 0.9, $x + y = o(k)$. After the first phase, the problem becomes that there are $s' = s - y = s - o(k)$ sites having bit 1, out of a total $k' = k - x - y = k - o(k)$ sites. The coordinator needs to figure out the exact value of s' with probability at least $0.8 - (1 - 0.9) = 0.7$.

In the second phase where all type (b) communication happens, from the coordinator's perspective, all the remaining sites are still symmetric (by the random input we choose), therefore the best it can do is to probe an arbitrary site among those that it has not communicated with. This is still true even after the coordinator has probed some of the remaining sites. Therefore, the problem boils down to the following: The coordinator picks z sites out of the remaining k' sites to communicate and then decides the value of s' with success probability at least 0.7. We call this problem the *sampling* problem. We can show that to achieve the success guarantee, z should be at least $\Omega(k)$. This result is perhaps folklore; proofs to more general versions of this problem can be found in [4] (Chapter 4), and also [22,26]. We include a simpler proof in the appendix for completeness. With this we conclude the proof of Lemma 2. $\square$

3 Tracking Distributed Frequencies

In the frequency-tracking problem, A (we omit "(t)" when the context is clear) is a multiset and the goal is to track the frequency of any item j within error εn. Let f_{ij} denote the local frequency of element j in A_i, and let $f_j = \sum_{i=1}^{k} f_{ij}$.

3.1 The Algorithm

The algorithm with a fixed p As in Sect. 2.1 we first describe the algorithm with a fixed parameter p. If each site tracks the local frequencies f_{ij} exactly, we can essentially use the count-tracking algorithm to track the f_j's. To achieve small space, we make use of the following algorithm due to Manku and Motwani [18] at each site S_i: The site maintains a list L_i of counters. When an element j arrives at S_i, the site first checks if there is a counter c_{ij} for j in L_i. If yes, it increases c_{ij} by 1. Otherwise, the site samples this element with probability p. If it is sampled, the site inserts a counter c_{ij}, initialized to 1, into L_i. It is easy to see that the expected size of L_i is $O(pn_i)$.

Next, we follow a similar strategy as in the count-tracking algorithm: The site reports the counter c_{ij} to the coordinator when it is first added to the counter list with an initial value of 1. Afterward, for every j that is arriving, the site always increments c_{ij} as before, but only sends the updated counter to the coordinator with probability p. We use $\bar{c}_{ij}$ to denote the last updated value of c_{ij}.

The tricky part is how the coordinator estimates f_{ij}, hence f_j. Fix any time instance. The difference between f_{ij} and $\hat{c}_{ij}$ comes from two sources: one is the number of j's missed before a copy is sampled, and the other is the number of j's that arrive after the last update of c_{ij}. It is easy to see that both errors follow the same distribution as $n_i - \bar{n}_i$ in the count-tracking algorithm. Thus it is tempting to modify (1) as

$$\hat{f}_{ij} = \begin{cases} \bar{c}_{ij} - 2 + 2/p, & \text{if } \bar{c}_{ij} \text{ exists;} \\ 0, & \text{else.} \end{cases} \tag{2}$$

However, this estimator is biased and its bias might be as large as $\Theta(\varepsilon n/\sqrt{k})$. Summing over k streams, this would exceed our error guarantee. To see this, consider the f_{ij} copies of j. Effectively, the site samples every copy with probability p, while $\bar{c}_{ij} - 2$ is exactly the number of copies between the first and the last sampled copy (excluding both). We define X_1 as before

$$X_1 = \begin{cases} t_1, & \text{if the } t_1\text{th copy is the first one sampled;} \\ f_{ij} + 1/p, & \text{if none is sampled.} \end{cases}$$

We define X_2 in exactly the same way, except that we examine these f_{ij} copies backward:

$$X_2 = \begin{cases} t_2, & \text{if the } t_2\text{th copy is the first one sampled} \\ & \text{in the reverse order;} \\ f_{ij} + 1/p, & \text{if none is sampled.} \end{cases}$$

It is clear that X_1 and X_2 have the same distribution with $\mathsf{E}[X_1] = \mathsf{E}[X_2] = 1/p$ (by Lemma 1), so $\hat{f}_{ij} = f_{ij} - (X_1 + X_2) + 2/p$ is unbiased. Since $\bar{c}_{ij} - 2 = f_{ij} - t_1 - t_2$, the correct unbiased estimator should be

$$\hat{f}_{ij} = \begin{cases} \bar{c}_{ij} - 2 + 2/p, & \text{if } \bar{c}_{ij} \text{ exists;} \\ -f_{ij}, & \text{else.} \end{cases} \tag{3}$$

Compared with the previous wrong estimator (2), the main difference is how the estimation is done when no copy of j is sampled. When $f_{ij} = \Theta(\varepsilon n/\sqrt{k})$ and $p = \Theta(1/f_{ij})$, with constant probability, none of j is sampled, which would result in a bias of $\Theta(f_{ij}) = \Theta(\varepsilon n/\sqrt{k})$.

However, the correct estimator (3) depends on f_{ij}, the quantity we want to estimate in the first place. The workaround is to use another unbiased estimator for f_{ij} when $\bar{c}_{ij}$ is not yet available. It turns out that we can just use simple random sampling: The site samples every element with probability p (this is independent of the sampling process that maintains the list L_i), and sends the sampled elements to the coordinator. Let d_{ij} be the number of sampled copies of j received by the coordinator from site i, the final estimator for f_{ij} is

$$\hat{f}'_{ij} = \begin{cases} \bar{c}_{ij} - 2 + 2/p, & \text{if } \bar{c}_{ij} \text{ exists;} \\ -d_{ij}/p, & \text{else.} \end{cases} \tag{4}$$

Since d_{ij} is independent of $\bar{c}_{ij}$, the estimator is still unbiased. Below we analyze its variance.

Analysis Intuitively, the variance is not affected by using the simple random sampling estimator d_{ij}/p, because it is only used when $\bar{c}_{ij}$ is not available, which means that f_{ij} is likely to be small, but when f_{ij} is small, d_{ij}/p actually has a small variance. When

f_{ij} is large, d_{ij}/p has a large variance, but we will use it only with small probability. Below we give a formal proof.

Lemma 3 $\mathsf{E}[\hat{f}'_{ij}] = f_{ij}$; $\mathsf{Var}[\hat{f}'_{ij}] = O(1/p^2)$.

Proof We first analyze the estimator $\hat{f}_{ij}$ of (3). That $\mathsf{E}[\hat{f}_{ij}] = f_{ij}$ follows from the discussion above. Its variance is $\mathsf{Var}[\hat{f}_{ij}] = \mathsf{Var}[X_1 + X_2]$. Note that X_1 and X_2 are not independent, but they both have expectation $1/p$ and variance $\le 1/p^2$. We first rewrite

$$\begin{aligned}\mathsf{Var}[X_1 + X_2] &= \mathsf{E}[X_1^2 + X_2^2 + 2X_1X_2] - \mathsf{E}[X_1 + X_2]^2\\ &= \mathsf{Var}[X_1] + \mathsf{E}[X_1]^2 + \mathsf{Var}[X_2] + \mathsf{E}[X_2]^2\\ &\quad + 2\mathsf{E}[X_1X_2] - (\mathsf{E}[X_1] + \mathsf{E}[X_2])^2\\ &\le 4/p^2 + 2\mathsf{E}[X_1X_2] - 4/p^2 \le 2\mathsf{E}[X_1X_2].\end{aligned}$$

Let $\mathcal{E}_t$ be the event that the tth copy of j is the first being sampled. We have

$$\begin{aligned}\mathsf{E}[X_1X_2] &= \sum_{t=1}^{f_{ij}} (1-p)^{t-1} pt\mathsf{E}[X_2 \mid \mathcal{E}_t] + (1-p)^{f_{ij}}(f_{ij} + 1/p)^2\\ &= \sum_{t=1}^{f_{ij}} (1-p)^{t-1} pt\left((1-p)^{f_{ij}-t}(f_{ij} - t + 1) + \sum_{l=1}^{f_{ij}-t} (1-p)^{l-1} pl\right)\\ &\quad +(1-p)^{f_{ij}}(f_{ij} + 1/p)^2\\ &\le \frac{1}{p^2} + (1-p)^{f_{ij}} f_{ij}^2 + \frac{(1-p)^{f_{ij}} f_{ij}}{p}.\end{aligned}$$

Let $c = f_{ij}p$. If $c \le 2$, i.e. $f_{ij} \le 2/p$, the variance is $O(1/p^2)$. If $c > 2$, we have

$$\mathsf{E}[X_1X_2] \le \frac{1}{p^2} + \frac{c^2}{p^2e^c} + \frac{c}{p^2e^c} = O(1/p^2),$$

since $c^2 \le e^c$ when $c > 2$.

Next we analyze the final estimator $\hat{f}'_{ij}$ of (4). First, d_{ij} is the sum of f_{ij} Bernoulli random variables with probability p, so $\mathsf{E}[d_{ij}/p] = f_{ij}$ and $\mathsf{Var}[d_{ij}/p] \le f_{ij}p/p^2 = f_{ij}/p$. Let $\mathcal{E}_*$ be the event that $\hat{c}_{ij}$ is available, i.e., at least one copy of j is sampled, and $\mathcal{E}_0 = \overline{\mathcal{E}_*}$, then

$$\begin{aligned}\mathsf{E}[\hat{f}'_{ij}] &= \mathsf{E}[\hat{f}_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] + \mathsf{E}[-d_{ij}/p \mid \mathcal{E}_0]\mathsf{Pr}[\mathcal{E}_0]\\ &= \mathsf{E}[\hat{f}_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] + (-f_{ij})\mathsf{Pr}[\mathcal{E}_0]\\ &= \mathsf{E}[\hat{f}_{ij}] = f_{ij}.\end{aligned}$$

The variance is

$$\begin{aligned}\mathsf{Var}[\hat{f}'_{ij}] &= \mathsf{E}[\hat{f}'^2_{ij}] - \mathsf{E}[\hat{f}'_{ij}]^2\\ &= \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] + \mathsf{E}[(d_{ij}/p)^2 \mid \mathcal{E}_0]\mathsf{Pr}[\mathcal{E}_0] - f^2_{ij}\\ &= \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] - f^2_{ij} + \mathsf{E}[(d_{ij}/p)^2]\mathsf{Pr}[\mathcal{E}_0]\\ &= \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] - f^2_{ij} + (\mathsf{Var}[d_{ij}/p] + f^2_{ij})\mathsf{Pr}[\mathcal{E}_0]\end{aligned}$$

Note that

$$\begin{aligned}\mathsf{Var}[\hat{f}_{ij}] &= \mathsf{E}[\hat{f}^2_{ij}] - f^2_{ij}\\ &= \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] + \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_0]\mathsf{Pr}[\mathcal{E}_0] - f^2_{ij}\\ &= \mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] + f^2_{ij}\mathsf{Pr}[\mathcal{E}_0] - f^2_{ij},\end{aligned}$$

hence $\mathsf{E}[\hat{f}^2_{ij} \mid \mathcal{E}_*]\mathsf{Pr}[\mathcal{E}_*] - f^2_{ij} = \mathsf{Var}[\hat{f}_{ij}] - f^2_{ij}\mathsf{Pr}[\mathcal{E}_0]$, which implies

$$\begin{aligned}\mathsf{Var}[\hat{f}'_{ij}] &= \mathsf{Var}[\hat{f}_{ij}] + \mathsf{Var}[d_{ij}/p]\mathsf{Pr}[\mathcal{E}_0]\\ &\le \mathsf{Var}[\hat{f}_{ij}] + \frac{f_{ij}}{p} \cdot (1-p)^{f_{ij}}.\end{aligned}$$

Due to the same reason as above, the second term is $O(1/p^2)$, and the proof completes. □

Dealing with a decreasing p As in the count-tracking algorithm, we divide the whole tracking period into $O(\log N)$ rounds. Within each round, n stays within a constant factor of $\bar{n}$, while $\bar{n}$ remains fixed for the whole round.

Within a round, we set the parameter p for all sites to be $p = 1/\lfloor \varepsilon\bar{n}/\sqrt{k} \rfloor_2$. When we proceed to a new round, all sites clear their memory and start a new copy of the algorithm from scratch with the new p. Given an item j, the coordinator estimates its frequency from each round separately, and add them up. Since the variance in a round is $O(k/p^2)$ and p decreases geometrically over the rounds, the total variance is asymptotically bounded by the variance of the last round, i.e., $O((\varepsilon n)^2)$, as desired.

The space used at some site could still be large, since the site may receive too many elements in a round. If all the $O(n)$ elements in a round have gone to the same site, the site will need to use space $O(pn) = O(\sqrt{k}/\varepsilon)$. To bound the space, we restrict the amount of space used by each site. More precisely, when a site receives more than $\bar{n}/k$ elements, it sends a message to the coordinator for notification, clears its memory, and starts a new copy of the algorithm from scratch. The coordinator will treat the new copy as if it were a new site, while the original site no longer receives more elements. Now the space used at each site is at most $p\bar{n}/k = O(1/(\varepsilon\sqrt{k}))$. Since there are at most $O(k)$ such new "virtual" sites ever created in a round, this does not affect the variance by more than a constant factor.

It remains to show that the total communication cost is $O(\sqrt{k}/\varepsilon \cdot \log N)$. From earlier we know that there are $O(\log N)$ rounds; within each round, $\bar{n}$ is the same and n stays within $\Theta(\bar{n})$. Focus on one round. For each arriving element, the site S_i updates $\bar{c}_{ij}$ with probability p and also independently samples it with probability p to maintain d_{ij}. This costs $O(n \cdot p) = O(\sqrt{k}/\varepsilon)$ communication.

Theorem 5 *There is a randomized algorithm for the frequency-tracking problem that tracks the frequency of any element within error εn with probability at least 0.9. It uses $O(1/(\varepsilon\sqrt{k}))$ space at each site and $O(\sqrt{k}/\varepsilon \cdot \log N)$ communication.*

3.2 Space Lower Bound

It is easy to see that the communication lower bounds for the count-tracking problem also hold for the frequency-tracking problem. In this section, we prove the following space-communication trade-off.

Theorem 6 *Consider any randomized algorithm for the frequency-tracking problem that, at any time, estimates the frequency of any element within error εn with probability at least 0.9. If the algorithm uses C bits of communication and uses M bits of space per site, then we must have $C \cdot M = \Omega(\log N/\varepsilon^2)$, assuming $k \le 1/\varepsilon^2$.*

Thus, if the communication cost is $C = O(\sqrt{k}/\varepsilon \cdot \log N)$ bits, the space required per site is at least $\Omega(1/(\varepsilon\sqrt{k}))$ bits, as claimed in Table 1. If we ignore the word/bit difference, the space bounds are also tight. Interestingly, this lower bound also shows that the random sampling algorithm [10] (see Table 1) actually attains the other end of this space-communication trade-off.

Proof (of Theorem 6) We will use a result in [27] for the one-shot version of the frequency estimation problem under the k-party communication model, i.e., all sites get all data in the beginning, and the goal is to estimate the global frequency of each element within error $2\varepsilon n$, where n is the total number elements. Their result states that, there is an input distribution μ_k such that, any algorithm that solves the one-shot version of the problem under μ_k with probability 0.9 needs at least $c\sqrt{k}/\varepsilon$ bits of communication for some constant c, assuming $k \le 1/\varepsilon^2$. Moreover, any algorithm that solves ℓ independent copies of the one-shot version simultaneously needs at least $\ell \cdot c\sqrt{k}/\varepsilon$ bits of communication.

We consider the frequency tracking problem over ρk sites, for some integer $\rho \ge 1$ to be determined later. The hard input distribution is defined as follow.

The input construction We divide the whole tracking period into $\log N$ rounds. In each round $i = 1, \ldots, \log N$, we first generate a (one-shot) input independently chosen from distribution $\mu_{\rho k}$, and for every element picked from $\mu_{\rho k}$ for any site, we duplicate it 2^{i-1} times. As a result, the number of elements in a round increases geometrically. In a round, site S_1 gets all its elements first, then S_2 gets all its elements, and so on so forth. We pick elements from a different domain for different rounds so that we have $\log N$ independent instances of the one-shot problem.

Let $\mathcal{A}_{\rho k}$ be any continuous tracking algorithm solving the frequency tracking problem over ρk sites. In our input construction, at any time, the last round always contains half of all the elements that have arrived so far, and thus $\mathcal{A}_{\rho k}$ must solve the one-shot problem that is "embedded" in each round, i.e., the algorithm must solve $\log N$ independent instances of the one-shot problem. By the result in [27], the communication cost of $\mathcal{A}_{\rho k}$ under the above input distribution is at least $c\sqrt{\rho k}/\varepsilon \cdot \log N$.

Let $\mathcal{A}_k$ be a continuous tracking algorithm over k sites that communicates C bits in total and uses M bits of space per site. Below we describe a ρk-site tracking algorithm for the input constructed above by simulating algorithm $\mathcal{A}_k$. The difficulty is that in our ρk-site problem, we have ρk sites receiving inputs, namely $S_1, \ldots, S_{\rho k}$, but $\mathcal{A}_k$ is an algorithm for only k sites, namely $S'_1, \ldots, S'_k$. However, note that in the above input construction, a site will receive all of its elements in a round before the next site starts to receive elements. Therefore, sites $S_1, \ldots, S_k$ can run $\mathcal{A}_k$ together before any element arrives at S_{k+1}; whenever $\mathcal{A}_k$ exchanges a message, they do the same. At some point, S_1 receives all its elements, then S_1 sends its memory content to S_{k+1} (relayed by the coordinator). With the memory footprint, S_{k+1} then takes the role of S_1 in the simulation of $\mathcal{A}_k$ and continues. The simulation is possible because S_1 will not receive any element during the rest of the round. Similarly, when S_2 has received all its elements, it sends its memory content to S_{k+2}, who replaces S_2 in the simulation. In general, when S_j is done with all its elements, it passes its role to S_{j+k}. When $S_{\rho k}$ is done, the simulation finishes for this round. $S_{\rho k}$ then sends a broadcast message and we proceed to the next round. It is clear that the above ρk-site algorithm solves the frequency tracking problem for the input constructed above as long as $\mathcal{A}_k$ is correct.

Let us analyze the communication cost of the simulation. First, all sites exchange exactly the same messages as $\mathcal{A}_k$ does, which costs C. We also communicate $\rho(k-1)$ memory snapshots and a broadcast message in each round, which costs $\leq \rho k M \log N$ over all rounds. Thus, we have

$$C + \rho k M \log N \geq c\sqrt{\rho k}/\varepsilon \cdot \log N.$$

After rearranging, we have

$$M \geq \frac{c}{\varepsilon\sqrt{\rho k}} - \frac{C}{\rho k \log N} = \frac{1}{\sqrt{\rho k}}\left(\frac{c}{\varepsilon} - \frac{C}{\sqrt{\rho k}\log N}\right).$$

Thus, if we set $\sqrt{\rho} = \left\lceil \frac{2C\varepsilon}{c\sqrt{k}\log N} \right\rceil$, then

$$M \geq \frac{c}{2\varepsilon\sqrt{\rho k}} = \Omega\left(\frac{\log N}{C\varepsilon^2}\right),$$

as claimed. □

4 Tracking Distributed Ranks

On a stream S of n elements, an algorithm that produces an unbiased estimator for any rank with variance $O((\varepsilon n)^2)$ was presented in [23], which has been very recently improved and made to work in a stronger model [1]. It uses $O(1/\varepsilon \cdot \log^{1.5}(1/\varepsilon))$

working space to maintain a rank estimation summary structure of size $O(1/\varepsilon)$. This summary can be used to produce an unbiased estimator for the rank (in S) of any query x with variance $O((\varepsilon n)^2)$. We call this algorithm $\mathcal{A}$ and will use it as a black box in our distributed tracking algorithm.

The overall algorithm. As before, with $O(k \log N)$ communication, we first track $\bar{n}$, a constant factor approximation of the current n. This also divides the tracking period into $O(\log N)$ rounds, and n increases by a constant factor during each rounds. At any time, each site runs an instance of algorithm $\mathcal{C}$, described below. The algorithm $\mathcal{C}$ will process a *chunk* of elements of size at most $\bar{n}/k$. A site may receive more than $\bar{n}/k$ elements in a round. When the $(\bar{n}/k + 1)$th element arrives, the site finishes the current instance of $\mathcal{C}$, and starts a new one, which will process the next $\bar{n}/k$ elements, and so on so forth. Therefore, the $\Theta(n)$ elements arriving in a round are divided into chunks of size at most $\bar{n}/k$, each of which is processed by an instance of algorithm $\mathcal{C}$. There are at most $2k$ chunks in a round, and thus, to estimate the rank of x in this round, it suffice to estimate the rank of x in each chunk with variance $O((\varepsilon n)^2/k)$.

Algorithm $\mathcal{C}$. Algorithm $\mathcal{C}$ reads at most $\bar{n}/k$ elements, and divides them into blocks of size $b = \varepsilon\bar{n}/\sqrt{k}$, so there are at most $\frac{1}{\epsilon\sqrt{k}}$ blocks. The algorithm builds a balanced binary tree on the blocks in the arrival order (see Fig. 1), and thus the height of the tree is $h \le \log \frac{1}{\epsilon\sqrt{k}}$. For each node v in the tree, let $D(v)$ be all the elements contained in the leaves in the subtree rooted at v. We say v is *full* if all the elements in $D(v)$ have arrived, and we say that v is *active* if some of the elements in $D(v)$ have arrived but v is still not full. For each active v, the site starts an instance of $\mathcal{A}$, denoted as $\mathcal{A}_v$, to process elements in $D(v)$ as they arrive. For a node v at level ℓ (the leaves are said to be on level 0), $|D(v)| = 2^\ell b$. We set the error parameter of $\mathcal{A}_v$ to $\varepsilon_\ell = 2^{-\ell}/\sqrt{h}$, so that the variance of the rank estimators is $O((\varepsilon_\ell|D(v)|))^2 = O(b^2/h)$. When v is full, the site sends the rank estimation summary computed by $\mathcal{A}_v$ to the coordinator, and free the space used by $\mathcal{A}_v$. The communication cost is the summary size, which is $O(1/\varepsilon_\ell) = O(\sqrt{h}2^\ell)$, while the space usage of $\mathcal{A}_v$ is $O(\sqrt{h}2^\ell \cdot \log^{1.5}(\sqrt{h}2^\ell))$.

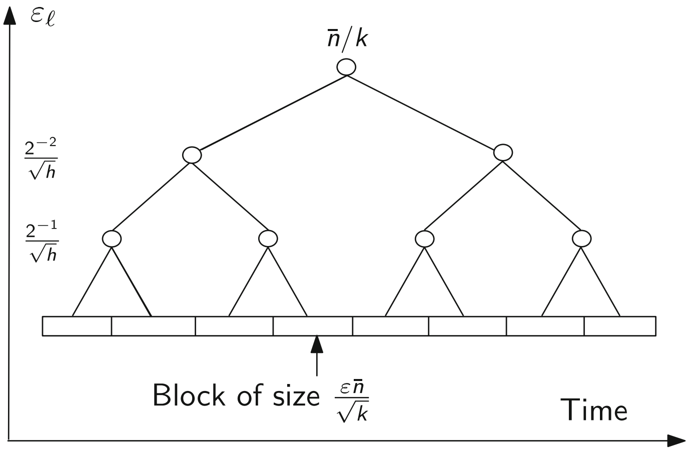
ε_ℓ
$\bar{n}/k$
$\frac{2^{-2}}{\sqrt{h}}$
$\frac{2^{-1}}{\sqrt{h}}$
Block of size $\frac{\varepsilon\bar{n}}{\sqrt{k}}$
Time

Fig. 1 Algorithm $\mathcal{C}$

Furthermore, for each new element, the site samples it with probability $p = \frac{\sqrt{k}}{\varepsilon \bar{n}}$, and if it is sampled, the site sends it to the coordinator.

Analysis of costs. We first analyze the space usage of $\mathcal{C}$. At any time, there are at most h active nodes, one at each level, so the space used by $\mathcal{C}$ is at most

$$\sum_{\ell=0}^{h} \sqrt{h} 2^{\ell} \log^{1.5} \frac{1}{\varepsilon\sqrt{k}} = O\left(\frac{\sqrt{h}}{\varepsilon\sqrt{k}} \log^{1.5} \frac{1}{\varepsilon\sqrt{k}}\right).$$

The communication for $\mathcal{C}$ includes all the summaries computed, and the elements sampled. For each node v on level ℓ, the size of the summary computed by $\mathcal{A}_v$ is $O(\sqrt{h}2^{\ell})$. There are $\frac{1}{\varepsilon\sqrt{k}} 2^{-\ell}$ nodes on level ℓ, hence the total size of the summaries on level ℓ is

$$O\left(\frac{1}{\varepsilon\sqrt{k}} 2^{-\ell} \cdot 2^{\ell}\sqrt{h}\right) = O\left(\frac{\sqrt{h}}{\varepsilon\sqrt{k}}\right).$$

Summing over all h levels, the total communication cost of $\mathcal{C}$ for sending summaries is $O(\frac{h^{1.5}}{\varepsilon\sqrt{k}})$. Since there are at most $2k$ instances of $\mathcal{C}$ in a round, the total communication cost in a round is $O(h^{1.5}\sqrt{k}/\varepsilon)$. The number of sampled elements in a round is $O(np) = O(\sqrt{k}/\varepsilon)$. Thus, over all $O(\log N)$ rounds, the total communication cost is $O(h^{1.5}\sqrt{k}/\varepsilon \cdot \log N)$.

Rank estimation. It remains to show how the coordinator estimates the rank of any given element x at any time with variance $O((\varepsilon n)^2)$, where n is the number of elements have arrived thus far. Based on the binary tree, we decompose all n elements into smaller disjoint subsets, and estimate the rank of x in each of the subsets. Since all estimators are unbiased, the overall estimator is also unbiased; the variance will be the sum of all the variances.

We will focus on the current round; all previous rounds can be handled similarly. Recall that there are $O(\bar{n})$ elements arriving in this round and $\bar{n} = \Theta(n)$, which are divided into chunks of $\bar{n}/k$ elements. Consider any such chunk processed by one instance of $\mathcal{C}$. Suppose up to now, $\mathcal{C}$ has processed n' elements in this chunk for some $n' \leq \bar{n}/k$. We write n' as $n' = q \cdot b + r$ for some $r < b$, and decompose these n' elements into at most $h + 1$ disjoint subsets as follows. The first qb elements are decomposed into at most h subsets, each of which corresponds to a full node in the binary tree of $\mathcal{C}$ (denoted as $v_1, \ldots, v_h$). Each v_i has already sent its summary (summarizing $D(v_i)$) to the coordinator, from which the coordinator can estimate the rank of any x in $D(v_i)$. Therefore, given x, we can estimate its rank in $\bigcup_i D(v_i)$. For each v_i on level ℓ, as we have discussed above, we can estimate the rank of x in $D(v_i)$ with variance is $((2^{-l}/\sqrt{h}) \cdot 2^l b)^2 = b^2/h$, so the total variance from all h nodes is b^2.

For the last r elements of the chunk that are still being processed by an active (leaf) node, the coordinator does not have any summary for them. But recall that the site always samples each element with probability $p = \sqrt{k}/(\varepsilon\bar{n})$ and sends it to the coordinator if it is sampled. Thus, the rank of x in these r elements can be estimated as follows: count the number of elements sampled which are smaller than x; rescale this count by $1/p$. It is quite standard to calculate the variance of this estimator, which is $r/p \le b/p = b^2$. Combined with the above analysis, we can estimate the rank of x in any chunk of $\bar{n}/k$ elements with variance $O(b^2)$. Since there are at most $2k$ such chunks in the round, the total variance is $O(b^2 k) = O((\varepsilon\bar{n})^2) = O((\varepsilon n)^2)$. As the variances of the previous rounds are geometrically decreasing, the total variance from all the rounds is still bounded by $O((\varepsilon n)^2)$, as desired.

Theorem 7 *There is a randomized algorithm for the rank-tracking problem that tracks the rank of any element within error εn with probability at least 0.9. The space used by each site is $O\left(\frac{1}{\varepsilon\sqrt{k}}\log^{1.5}\frac{1}{\varepsilon}\log^{0.5}\frac{1}{\varepsilon\sqrt{k}}\right)$ and the total communication cost is $O\left(\frac{\sqrt{k}}{\varepsilon}\log N \log^{1.5}\frac{1}{\epsilon\sqrt{k}}\right)$.*

Appendix: Lower Bound for the Sampling Problem

Claim To solve the sampling problem we need to probe at least $\Omega(k)$ sites.

Proof Suppose that the coordinator only samples $z = o(k)$ sites. Let X be the number of sites that are sampled with bit 1. Then X is chosen from the hypergeometric distribution with probability density function (pdf) $\Pr[X = x] = \binom{s'}{x}\binom{k'-s'}{z-x}/\binom{k'}{z}$. The expected value of X is $\frac{z}{k'} \cdot s'$, which is $\frac{z}{k'}\left(\frac{k}{2} - y + \sqrt{k}\right)$ or $\frac{z}{k'}\left(\frac{k}{2} - y - \sqrt{k}\right)$, depending on the value of s'. Let $p = \left(\frac{k}{2} - y\right)/k' = \frac{1}{2} \pm o(1)$ and $\alpha = \sqrt{k}/k' = 1/\sqrt{k} \pm o(1/\sqrt{k})$. To avoid tedious calculation, we assume that X is picked randomly from one of the two normal distributions $\mathcal{N}_1(\mu_1, \sigma_1^2)$ and $\mathcal{N}_2(\mu_2, \sigma_2^2)$ with equal probability, where $\mu_1 = z(p - \alpha)$, $\mu_2 = z(p + \alpha)$, $\sigma_1, \sigma_2 = \Theta(\sqrt{zp(1-p)}) = \Theta(\sqrt{z})$. In Feller [11] it is shown that the normal distribution approximates the hypergeometric distribution very well when z is large and $p \pm \alpha$ are constants in $(0, 1)$.[5] Now our task is to decide from which of the two distributions X is drawn based on the value of X with success probability at least 0.7.

Let $f_1(x; \mu_1, \sigma_1^2)$ and $f_2(x; \mu_2, \sigma_2^2)$ be the pdf of the two normal distributions $\mathcal{N}_1, \mathcal{N}_2$, respectively. It is easy to see that the best deterministic algorithm of differentiating the two distributions based on the value of a sample X will do the following.

- If $X > x_0$, then X is chosen from $\mathcal{N}_2$, otherwise X is chosen from $\mathcal{N}_1$, where x_0 is the value such that $f_1(x_0; \mu_1, \sigma_1^2) = f_2(x_0; \mu_2, \sigma_2^2)$ (thus $\mu_1 < x_0 < \mu_2$).

Indeed, if $X > x_0$ and the algorithm decides that "X is chosen from $\mathcal{N}_1$", we can always flip this decision and improve the success probability of the algorithm.

[5] In Feller's book [11] the following is proved. Let $p \in (0, 1)$ be some constant and $q = 1 - p$. The population size is N and the sample size is n, so that $n < N$ and Np, Nq are both integers. The hypergeometric distribution is $P(k; n, N) = \binom{Np}{k}\binom{Nq}{n-k}/\binom{N}{n}$ for $0 \le k \le n$.

Theorem 8 [11] *If $N \to \infty$, $n \to \infty$ so that $n/N \to t \in (0, 1)$ and $x_k := (k - np)/\sqrt{npq} \to x$, then*

$$p(k; n, N) \sim \frac{e^{-x^2/2(1-t)}}{\sqrt{2\pi npq(1-t)}}$$

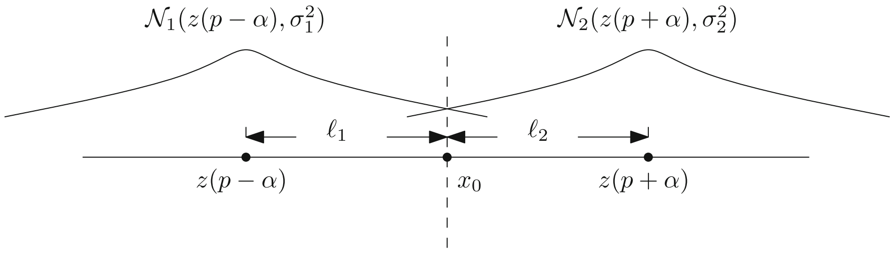
$\mathcal{N}_1(z(p-\alpha),\sigma_1^2)$
$\mathcal{N}_2(z(p+\alpha),\sigma_2^2)$
ℓ_1
ℓ_2
$z(p-\alpha)$
x_0
$z(p+\alpha)$

Fig. 2 Differentiating two distributions

The error comes from two sources: (1) $X > x_0$ but X is actually drawn from $\mathcal{N}_2$; (2) $X \le x_0$ but X is actually drawn from $\mathcal{N}_1$. The total error is

$$1/2 \cdot (\Phi(-\ell_1/\sigma_1) + \Phi(-\ell_2/\sigma_2)),$$

where $\ell_1 = x_0 - \mu_1$ and $\ell_2 = \mu_2 - x_0$. (Thus $\ell_1 + \ell_2 = \mu_2 - \mu_1 = 2\alpha z$). $\Phi(\cdot)$ is the cumulative distribution function (cdf) of the normal distribution. See Fig. 2.

Finally note that $\ell_1/\sigma_1 = O(\alpha z/\sqrt{z}) = O(\sqrt{z/k}) = o(1)$ and $\ell_2/\sigma_2 = O(\alpha z/\sqrt{z}) = o(1)$, so $\Phi(-\ell_1/\sigma_1) + \Phi(-\ell_2/\sigma_2) > 0.99$. Therefore, the failure probability is at least 0.49, contradicting our success probability guarantee. Thus we must have $z = \Omega(k)$. □

References

1. Agarwal, P.K., Cormode, G., Huang, Z., Phillips, J.M., Wei, Z., Yi, K.: Mergeable summaries. In: Proceedings of the ACM Symposium on Principles of Database Systems (2012)
2. Arackaparambil, C., Brody, J., Chakrabarti, A.: Functional monitoring without monotonicity. In: Proceedings of the International Colloquium on Automata, Languages, and Programming (2009)
3. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2003)
4. Bar-Yossef, Z.: The complexity of massive data set computations. PhD thesis, University of California at Berkeley (2002)
5. Chan, H.-L., Lam, T.W., Lee, L.-K., Ting, H.-F.: Continuous monitoring of distributed data streams over a time-based sliding window. Algorithmica **62**(3–4), 1088–1111 (2011)
6. Cormode, G.: The continuous distributed monitoring model. ACM SIGMOD Rec. **42**(1), 5–14 (2013)
7. Cormode, G., Garofalakis, M., Muthukrishnan, S., Rastogi, R.: Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2005)
8. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. In: Proceedings of the International Conference on Very Large Data Bases (2008)
9. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. ACM Trans. Algorithms **7**(2), Article 21 (2011). (Preliminary version in SODA'08)
10. Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Continuous sampling from distributed streams. J. ACM **59**(2), 10 (2012). (Preliminary version in PODS'10)
11. Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley, New York (1968)
12. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (2001)
13. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2001)

14. Huang, Z., Wang, L., Yi, K., Liu, Y.: Sampling based algorithms for quantile computation in sensor networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2011)
15. Huang, Z., Yi, K., Liu, Y., Chen, G.: Optimal sampling algorithms for frequency estimation in distributed data. In: IEEE INFOCOM (2011)
16. Keralapura, R., Cormode, G., Ramamirtham, J.: Communication-efficient distributed monitoring of thresholded counts. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2006)
17. Manjhi, A., Shkapenyuk, V., Dhamdhere, K., Olston, C.: Finding (recently) frequent items in distributed data streams. In: Proceedings of the IEEE International Conference on Data Engineering (2005)
18. Manku, G., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the International Conference on Very Large Data Bases (2002)
19. Metwally, A., Agrawal, D., Abbadi, A.: An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans. Database Syst. **31**(3), 1095–1133 (2006)
20. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Program. **2**, 143–152 (1982)
21. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. Theor. Comput. Sci. **12**, 315–323 (1980)
22. Patt-Shamir, B., Shafrir, A.: Approximate distributed top-k queries. Distrib. Comput. **21**(1), 1–22 (2008)
23. Suri, S., Toth, C., Zhou, Y.: Range counting over multidimensional data streams. Discrete Comput. Geom. **36**, 633–655 (2006)
24. Tirthapura, S., Woodruff, D.P.: Optimal random sampling from distributed streams revisited. In: Proceedings of the International Symposium on Distributed Computing (2011)
25. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Appl. **16**, 264–280 (1971)
26. Woodruff, D.P.: Efficient and Private Distance Approximation in the Communication and Streaming Models. PhD thesis, Massachusetts Institute of Technology (2007)
27. Woodruff, D.P., Zhang, Q.: Tight bounds for distributed functional monitoring. In: Proceedings of the ACM Symposium on Theory of Computing (2012)
28. Yao, A.C.: Probabilistic computations: towards a unified measure of complexity. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (1977)
29. Yi, K., Zhang, Q.: Optimal tracking of distributed heavy hitters and quantiles. In: Proceedings of the ACM Symposium on Principles of Database Systems (2009)