Tracking Distributed Aggregates over Time-Based Sliding Windows*

Graham Cormode[1] and Ke Yi[2,**]

[1] AT&T Labs–Research
graham@research.att.com
[2] Hong Kong University of Science and Technology
yike@cse.ust.hk

Abstract. The area of distributed monitoring requires tracking the value of a function of distributed data as new observations are made. An important case is when attention is restricted to only a recent time period, such as the last hour of readings—the sliding window case. In this paper, we introduce a novel paradigm for handling such monitoring problems, which we dub the "forward/backward" approach. This view allows us to provide optimal or near-optimal solutions for several fundamental problems, such as counting, tracking frequent items, and maintaining order statistics. The resulting protocols improve on previous work or give the first solutions for some problems, and operate efficiently in terms of space and time needed. Specifically, we obtain optimal $O(\frac{k}{\varepsilon}\log(\varepsilon n/k))$ communication per window of n updates for tracking counts and heavy hitters with accuracy ε across k sites; and near-optimal communication of $O(\frac{k}{\varepsilon}\log^2(1/\varepsilon)\log(n/k))$ for quantiles. We also present solutions for problems such as tracking distinct items, entropy, and convex hull and diameter of point sets.

1 Introduction

Problems of distributed tracking involve trying to compute various aggregates over data that is distributed across multiple observing sites. Each site observes a stream of information, and aims to work together with the other sites to continuously track a function over the union of the streams. Such problems arise in a variety of modern data management and processing settings—for more details and motivating examples, see the recent survey of this area [6]. To pick one concrete example, a number of routers in a network might try to collaborate to identify the current most popular destinations. The goal is to allow a single distinguished entity, known as the "coordinator", to track the desired function. Within such settings, it is natural to only want to capture the recent behavior—say, the most popular destinations within the last 24 hours. Thus, attention is limited to a "time-based sliding window".

* These results were announced at PODC'11 as a 'brief announcement', with an accompanying 2 page summary.

** Ke Yi is supported by an RPC grant from HKUST and a Google Faculty Research Award.

A. Ailamaki and S. Bowers (Eds.): SSDBM 2012, LNCS 7338, pp. 416–430, 2012.

For these problems, the primary goal is to minimize the (total) communication required to achieve accurate tracking. Prior work has shown that in many cases this cost is asymptotically smaller than the trivial solution of simply centralizing all the observations at the coordinator site. Secondary goals include minimizing the space required at each site to run the protocol, and the time to process each new observation. These quantities are functions of k, the number of distributed sites, n, the total size of the input data, and ε, an a user-supplied approximation parameter to tolerate some imprecision in the computed answer (typically, $0 < \varepsilon < 1$).

Within this context, there has been significant focus on the "infinite window" case, where all historic data is included. Results have been shown for monitoring functions such as counts, distinct counts, order statistics, join sizes, entropy, and others [1,4,7,14,19,20]. More recently there has been interest in only tracking a window of recent observations, defined by all those elements which arrived within the most recent w time units. Results in this model have been shown for tracking counts and frequent items [4], and for sampling [8].

The most pertinent prior work is that of Chan *et al.* [4], which established protocols for several fundamental problems with sliding windows. The analysis used quickly becomes quite complicated, due to the need to study multiple cases in detail as the distributions change. Perhaps due to this difficulty, the bounds obtained are not always optimal. Three core problems are studied: basic counting, which is to maintain the count of items observed within the window; heavy hitters, which is to maintain all items whose frequency (within the window) is more than a given fraction; and to maintain the quantiles of the distribution. Each problem tolerates an error of ε, and is parametrized by k, the number of sites participating in the computation, and n, the number of items arriving in a window. [4] shows (per window) communication costs of $O(\frac{k}{\varepsilon} \log \frac{\varepsilon n}{k})$ bits for basic counting, $O(\frac{k}{\varepsilon} \log \frac{n}{k})$ words[1] for frequent items and $O(\frac{k}{\varepsilon^2} \log \frac{n}{k})$ words for quantiles. Our main contributions in this paper are natural protocols with a more direct analysis which obtain optimal or near optimal communication costs. To do this, we outline an approach for decomposing sliding windows, which also extends naturally to other problems in this setting. We call this the "forward/backward" framework, and provide a general claim, that the communication complexity for many functions in the model with a sliding window is no more than in the infinite window case (Section 2). We instantiate this to tracking counts (Section 3), heavy hitters (Section 4) and quantiles (Section 5) to obtain optimal or near optimal communication bounds, with low space and time costs. Lastly, we extend our results to functions which have not been studied in the sliding window model before, such as distinct counts, entropy, and geometric properties in Section 6.

Other Related Work. Much of the previous work relies on monotonic properties of the function being monitored to provide cost guarantees. For example, since a count (over an infinite window) is always growing, the cost of most approximate tracking algorithms grows only logarithmically with the number of updates [7]. But the adoption of a time-based sliding window can make a

[1] Here, words is shorthand for machine words, in the standard RAM model.

previously monotonic function non-monotonic. That is, a function which is monotonic over an infinite window (such as a count) can decrease over a time-based window, due to the implicit deletions. Sharfman *et al.* [19] gave a generic method for arbitrary functions, based on a geometric view of the input space. This approach relies on keeping full space at each monitoring site, and does not obviously extend to functions which do not map on to single values (such as heavy hitters and quantiles). Arackaparambil *et al.* [1] study (empirical) entropy, which is non-monotonic. The protocols rely on a slow changing property of entropy: a constant change in the value requires a constant factor increase in the number of observations, which keeps the communication cost logarithmic in the size of the input. This slow-changing property does not hold for general functions. Distributed sliding window computations have also received much attention in the non-continuous-tracking case [3,11], where the goal is to keep a small amount of information over the stream at each site, so that the desired aggregate can be computed upon request; here, we have the additional challenge of tracking the aggregate at all times with small communication.

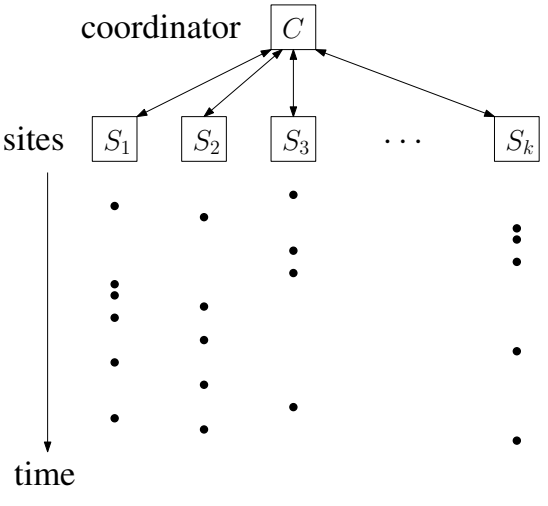

Fig. 1. Schematic of the distribute streaming model

1.1 Problem Definitions and Our Results

Now we more formally define the problems studied in this paper. Figure 1 shows the model schematically: k sites each observe a stream S_i of item arrivals, and communicate with a single distinguished coordinator node to continuously compute some function of the union of the update streams.

The *basic counting* problem is to track (approximately) the number of items which have arrived across all sites within the last w time units. More precisely, let the stream of items observed at site i be S_i, a set of $(x, t(x))$ pairs, which

indicates that an item x arrives at time $t(x)$. Then the exact basic count at time t is given by

$$C(t) = \sum_{1 \le i \le k} |\{(x, t(x)) \in S_i \mid t - t(x) \le w\}|.$$

Tracking $C(t)$ exactly requires alerting the coordinator every time an item arrives or expires, so the goal is to track $C(t)$ approximately within an ε-error, i.e., the coordinator should maintain a $\tilde{C}(t)$ such that $(1-\varepsilon)C(t) \le \tilde{C}(t) \le (1+\varepsilon)C(t)$ at all times t. We will assume that at each site, at most one item arrives in one time unit. This is not a restriction, since we can always subdivide the time units into smaller pieces so that at most one item arrives within one unit. This rescales w but does not fundamentally change our results, since the bounds provided do not depend on w.

The *heavy hitters* problem extends the basic counting problem, and generalizes the concept of finding the mode [15]. In the basic counting problem we count the total number of all items, while here we count the frequency of every distinct item x, i.e., the coordinator tracks the approximate value of

$$n_x(t) = \sum_{1 \le i \le k} |(x, t(x)) \in S_i \mid t - t(x) \le w\}|.$$

Since it is possible that many $n_x(t)$ are small, say 0 or 1 for all x, requiring a multiplicative approximation for all x would require reporting all items to the coordinator. Consequently, the commonly adopted approximation guarantee for heavy hitters is to maintain a $\tilde{n_x}(t)$ that has an additive error of at most $\varepsilon C(t)$, where $C(t)$ is the total count of all items. This essentially makes sure that the "heavy" items are counted accurately while compromising on the accuracy for the less frequent items. In particular, all items x with $n_x(t) \le \varepsilon C(t)$ can be ignored altogether as 0 is considered a good approximation for their counts.[2] This way, at most $1/\varepsilon$ distinct items will have nonzero approximated counts.

The *quantiles* problem is to continuously maintain approximate order statistics on the distribution of the items. That is, the items are drawn from a total order, and we wish to retain a set of items $q_1, \ldots, q_{1/\varepsilon}$ such that the rank of q_i (number of input items within the sliding window that are less than q_i) is between $(i-1)\varepsilon C(t)$ and $(i+1)\varepsilon C(t)$ [18]. It is known that this is equivalent to the "prefix-count" problem, where the goal is to maintain a data structure on the sliding window such that for any given x, the number of items smaller than x can be counted within an additive error of at most $\varepsilon C(t)$.

Figure 2 summarizes our main results. The communication cost is measured as the total amount of communication between all k sites and the central coordinator site, as a function of n, the number of observations in each window, and ε, the approximation parameter. All communication costs are optimal or near-optimal up to polylogarithmic factors. We also list the space required by each site to run the protocol.

[2] We may subsequently drop the (t) notation on variables when it is clear from the context.

Problem	Communication Cost	Communication lower bound	Space Cost
Basic Counting	$O(\frac{k}{\varepsilon}\log(\varepsilon n/k))$ bits	$\Omega(\frac{k}{\varepsilon}\log(\varepsilon n/k))$ bits	$O(\frac{1}{\varepsilon}\log \varepsilon n)$
Heavy Hitters	$O(\frac{k}{\varepsilon}\log(\varepsilon n/k))$	$\Omega(\frac{k}{\varepsilon}\log(\varepsilon n/k))$ bits	$O(\frac{1}{\varepsilon}\log \varepsilon n)$
Quantiles	$O(\frac{k}{\varepsilon}\log^2(1/\varepsilon)\log(n/k))$	$\Omega(\frac{k}{\varepsilon}\log(\varepsilon n/k))$ bits	$O(\frac{1}{\varepsilon}\log^2(1/\varepsilon)\log n)$

Fig. 2. Summary of Results. All bounds are in terms of words unless specified otherwise.

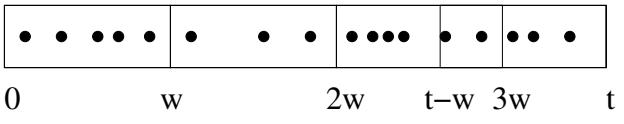

Fig. 3. Item arrivals within fixed windows

2 The Forward/Backward Framework

To introduce our framework, we observe that the problems defined in Section 1.1 all tolerate an error of $\varepsilon C(t)$, where $C(t)$ is the total number of items in the sliding window from all k sites. If we can track the desired count for every site within an error of $\varepsilon C^{(i)}(t)$, where $C^{(i)}(t)$ is the number of items at site i in the sliding window, then the total error will be $\sum_{i=1}^{k} \varepsilon C^{(i)}(t) = \varepsilon C(t)$. So we can focus on accurately tracking the data of one site, and combine the results of all sites to get the overall result.

Next, assuming the time axis starts at 0, we divide it into fixed windows of length w: $[0, w)$, $[w, 2w)$, ..., $[jw, (j+1)w), \ldots$. Then at at time t, the sliding window $[t-w, t)$ overlaps with at most two of these fixed windows, say $[(j-1)w, jw)$ and $[jw, (j+1)w)$. This splits the sliding window into two smaller windows: $[t-w, jw)$ and $[jw, t)$. We call the first one the *expiring window* and the second the *active window*. Figure 3 shows this schematically: item arrivals, shown as dots, are partitioned into fixed windows. At the current time, t, which in this example is between $3w$ and $4w$, it induces the expiring window $[t-w, 3w)$ (with two items in the example) and the active window $[3w, t)$ (with a further three items). As the window $[t-w, t)$ slides, items expire from the expiring window, while new items arrive in the active window. The problem is again decomposed into tracking the desired function in these two windows, respectively. Care must be taken to ensure that the error in the approximated count is with respect to the number of items in the active (or expiring) window, not that of the fixed window. However, a key simplification has happened: now (with respect to the fixed time point jw), the counts of items in the expiring window are only decreasing, while the counts of items in the active window are only increasing. As a result we make an (informal) claim about the problem:

Claim. For tracking a function in the sliding window continuous monitoring setting, the asymptotic communication complexity per window is that of the infinite window case.

To see this, observe that, using the above simplification, we now face two subproblems: (i) forward: tracking the active window and (ii) backward: tracking the expiring window. Tracking the active window is essentially the same as the infinite window case, hence the cost (per window) is that of running a protocol for the infinite window case. However, for the expiring window we also face essentially the same problem: we need a protocol which ensures that the coordinator always knows a good approximation to the function for the window as items expire instead of arrive. When we view this in the reverse time direction, the expirations become arrivals. If we ran the infinite window protocol on this time-reversed input, we would meet the requirements for approximating the function. Therefore, we can take the messages that are sent by this protocol, and send them all to the coordinator in one batch at the end of each fixed window. The site can send a bit to the coordinator at each time step when it would have sent the next message (in the time-reversed setting). Thus, the coordinator always has the state in the forward direction that it would have had in the time-reversed direction. □

This outline requires the site to record the stream within each window so it can perform this time-reversal trick. The problem gets more challenging if we also require small space at the site. For this we adapt small-space sliding window algorithms from the streaming literature to compactly "encode" the history. Next we show how to instantiate the forward/backward framework in a space efficient way for each of the three problems defined earlier. The forward problem (i.e., the full stream case) has been studied in prior work (for example, [20] gave results for heavy hitters and quantiles), but we are able to present simpler algorithms here. The lower bounds for the forward problem apply to the forward/backward case, and so we are able to confirm that our solutions are optimal or near-optimal.

3 Warm-Up: Basic Counting

The Forward Problem. For basic counting, the forward problem is to track the number of items that have arrived since a *landmark* t_0, up to a multiplicative error of $(1+\varepsilon)$. This is straightforward: the site simply sends a bit every time this number has increased by a $(1+\varepsilon)$ factor. This results in a communication cost of $O(1/\varepsilon \cdot \log n^{(i)})$ bits, where $n^{(i)}$ is the number of items in the fixed window at the site i when this forward tracking problem takes place. This can be tightened to $O(1/\varepsilon \cdot \log(\varepsilon n^{(i)}))$ by observing that the site actually sends out $1/\varepsilon$ bits for the first $1/\varepsilon$ items. Summing over all k sites and using the concavity of the log function gives $O(\frac{k}{\varepsilon} \log \frac{\varepsilon n}{k})$, matching the bound of [4]. The space required is minimal, just $O(1)$ for each site to track the current count in the active window.

The Backward Problem. As noted above, if we can buffer all the input, then we can solve the problem by storing the input, and compute messages based on item expiries. To solve the backward problem with small space (without buffering the current window) is more challenging. To do so, we make use of the *exponential histogram* introduced in [9]. Let the active window be $[t_0, t)$. Besides running the forward algorithm from t_0, each site also maintains an exponential histogram

starting from t_0. It records the ε^{-1} most recent items (and their timestamps), then every other item for another stored ε^{-1} items, then every fourth item, and so on. This is easily maintained as new items arrive: when there are more than $\varepsilon^{-1}+1$ items at a particular granularity, the oldest two can be "merged" into a single item at the next coarser granularity. Let t be the current time. When $t = t_0 + w$, the site freezes the exponential histogram. At this time, we set $t_0 \leftarrow t_0 + w$, and the active window becomes the expiring window while a new active window starts afresh. It follows from this description that the size of the exponential histogram is $O(\varepsilon^{-1}\log(\varepsilon n^{(i)}))$.

With an exponential histogram for the window $[t_0 - w, t_0)$, one can approximately count the items in the interval $[t - w, t_0)$, i.e., the expiring window at time t. We find in the exponential histogram two adjacent timestamps t_1, t_2 such that $t_1 < t - w \leq t_2$. Note that from the data structure we can compute the number of items in the time interval $[t_2, t_0)$ exactly, which we use as an estimate for the number of items in $[t - w, t_0)$. This in the worst case will miss all items between t_1 and t_2, and there are 2^a of them for some a. The construction of the exponential histogram ensures that $C_i(t_2, t_0) \geq \varepsilon^{-1}2^a$, where $C_i(t_2, t_0)$ denotes the number of items that arrived between time t_2 and t_0. So the error is at most $\varepsilon C_i(t_2, t_0) \leq \varepsilon C_i(t - w, t_0)$, as desired.

There are two ways to use the exponential histogram in a protocol. Most directly, each site can send its exponential histogram summarizing $[t_0 - w, t_0)$ to the coordinator at time t_0. From these, the coordinator can approximate the total count of the expiring window accurately. However, this requires the coordinator to store all k windows, and is not communication optimal. Instead, the space and communication cost can be reduced by having each site retain its exponential histogram locally. At time t_0, each site informs the coordinator of the total number of timestamps stored in its histogram of $[t_0 - w, t_0)$. Then each site sends a bit to the coordinator whenever any timestamp recorded in the histogram expires (i.e., reaches age w). This information is sufficient for the coordinator to recreate the current approximate count of the expiring window for each site. The communication cost is the same as the forward case, i.e., $O(1/\varepsilon \cdot \log(\varepsilon n^{(i)}))$ bits.

Theorem 1. *The above protocol for basic counting has a total communication cost of* $O(\frac{1}{\varepsilon}\log(\varepsilon n^{(i)}))$ *bits for each site, implying a total communication cost of* $O(\frac{k}{\varepsilon}\log\frac{\varepsilon n}{k})$ *per window. The space required at each site is* $O(\frac{1}{\varepsilon}\log(\varepsilon n^{(i)}))$ *words, and* $O(k)$ *at the coordinator to keep track of the current estimates from each site.*

This bound is optimal: the lower bound for an infinite window is $\Omega(\frac{k}{\varepsilon}\log\frac{\varepsilon n}{k})$ bits of communication. We see the power of the forward/backward framework: the analysis matches the bound in [4], but is much more direct. The dependence on $O(\log n)$ is unavoidable, as shown by the lower bounds. However, note that we do not require explicit knowledge of n (or an upper bound on it). Rather, the communication used, and the space requires, scales automatically with $\log n$, as the stream unfolds.

Algorithm 1. HeavyHittersArrivals

```
1 ∀x, n_x^(i) = 0 ;
2 A^(i) = 1;
3 foreach arrival of x do
4     n_x^(i) ← n_x^(i) + 1 ;
5     n^(i) ← n^(i) + 1 ;
6     if n_x^(i) mod A^(i) = 0 then
7         Send (x, n_x^(i)) to coordinator;
8     if n^(i) ≥ 2ε^{-1} A^(i) then
9         A^(i) ← 2A^(i);
```

4 Heavy Hitters

The Forward Problem. For simplicity, we first present an algorithm for the forward problem that assumes that each site has sufficient space to retain all "active" (non-expired) items locally. Then we discuss how to implement the algorithm in less space below.

Starting from time t_0, each site executes HeavyHitterArrivals (Algorithm 1) on the newly arriving items until $t = t_0 + w$. This tracks counts of each item in the active window ($n_x^{(i)}$ for the count of item x at site i), and ensures that the coordinator knows the identity and approximate count of any item with an additive error of at most $A^{(i)} - 1$. Note that $A^{(i)}$ is always at most $\varepsilon n^{(i)}$, where $n^{(i)}$ is the total number of items in the active window at site i, so correctness follows easily from the algorithm.

We next bound the communication cost. While $n^{(i)}$ is between $2^a\varepsilon^{-1}$ and $2^{a+1}\varepsilon^{-1}$, $A^{(i)} = 2^a$. For each distinct x, line 7 of the algorithm is called whenever $A^{(i)}$ new copies of x have arrived, except possibly the first call. Ignoring the first call to every distinct x, line 7 is executed at most $2^a\varepsilon^{-1}/A^{(i)} = \varepsilon^{-1}$ times. Note that for an item x to trigger line 7 at least once, $n_x^{(i)}$ has to be at least $A^{(i)}$, and there are at most $2^{a+1}\varepsilon^{-1}/A^{(i)} = O(\varepsilon^{-1})$ such distinct items, so the number of first calls is at most $O(\varepsilon^{-1})$. Hence, the total amount of information sent during this phase of the algorithm is $O(\varepsilon^{-1})$ items and counts. In total, there are $O(\log(\varepsilon n^{(i)}))$ such phases corresponding to the doubling values of $A^{(i)}$, and the total communication cost is $O(\varepsilon^{-1}\log(\varepsilon n^{(i)}))$. Summed over all sites the protocol costs $O(k\varepsilon^{-1}\log(\varepsilon n/k))$ per window.

The Backward Problem. In the case where we can afford to retain all the stream arrivals during the current window, we use a similar algorithm to solve the backward problem. Each site executes HeavyHitterExpiries in parallel on the expiring window (Algorithm 2). Conceptually, it is similar to running the previous algorithm in reverse. It maintains a parameter $B^{(i)}$ which denotes the local tolerance for error. The initial value of $B^{(i)}$ is equivalent to the final value

Algorithm 2. HeavyHittersExpiries

```
1  Send n^(i) to the coordinator;
2  B^(i) ← 2^⌊log εn^(i)⌋ ;
3  while n^(i) > 0 do
4      foreach x do
5          if n_x^(i) ≥ B^(i) then send x, n_x^(i)
6      while n_x^(i) > ε^{-1} B^(i) or (B^(i) ≤ 1 and n^(i) > 0) do
7          foreach expiry of x do
8              n^(i) ← n^(i) − 1 ;
9              n^(i) ← n_x^(i) − 1;
10             if n_x^(i) mod B^(i) = 0 then
11                 Send x, n_x^(i) to the coordinator
12     B^(i) ← B^(i)/2 ;
```

of $A^{(i)}$ from the active window which has just concluded. Letting $n^{(i)}$ denote the number of items from $[t-w, t_0)$ (i.e., those from the expiring window which have not yet expired), $B^{(i)}$ remains in the range $[\frac{1}{2}\varepsilon n^{(i)}, \varepsilon n^{(i)}]$. Whenever $B^{(i)}$ is updated by halving, the algorithm sends all items and counts where the local count is at least $B^{(i)}$. Since $B^{(i)}$ is $O(\varepsilon n)$, there are $O(\varepsilon^{-1})$ such items to send. As in the forward case, these guarantees ensure that the accuracy requirements are met.

The communication cost is bounded similarly to the forward case. There are $\log(\varepsilon n^{(i)})$ iterations of the outer loop until $B^{(i)}$ reaches 1. In each iteration, there are $O(\frac{1}{\varepsilon})$ items sent in line 5 that exceed $B^{(i)}$. Then at most $O(\frac{1}{\varepsilon})$ updates can be sent by line 11 before $B^{(i)}$ decreases. When $B^{(i)}$ reaches 1, there are only $1/\varepsilon$ unexpired items, and information on these is sent as each instance expires. This gives a total cost of $O(\frac{1}{\varepsilon}\log(\varepsilon n^{(i)}))$, which is $O(\frac{k}{\varepsilon}\log(\varepsilon n))$ when summed over all k sites.

At any time, the coordinator has information about a subset of items from each site (from both the active and expiring windows). To estimate the count of any item, it adds all the current counts for that item together. The error bounds ensure that the total error for this count is at most εn. To extract the heavy hitters, the coordinator can compare the estimated counts to the current (estimated) value of n, computed by a parallel invocation of the above basic counting protocol.

Reducing the Space Needed at Each Site. To reduce space used at the site for the forward problem, it suffices to replace the exact tracking of all arriving items with a small space algorithm to approximate the counts of items. For example, the SpaceSaving algorithm [17] tracks $O(1/\varepsilon)$ items and counts, so that item frequencies are reported with error at most $\varepsilon n^{(i)}$. This adds another $\varepsilon n^{(i)}$ to the error at the coordinator side, making it $2\varepsilon n^{(i)}$, but a rescaling of ε suffices

to bring it back to $\varepsilon n^{(i)}$. The communication cost does not alter: by the guarantee of the algorithm, there can still be only $O(\varepsilon^{-1})$ items whose approximate count exceeds $A^{(i)}$. While these items exceed $A^{(i)}$, their approximate counts are monotone increasing, so the number of messages does not increase.

For the backward part, the details are slightly more involved. We require an algorithm for tracking approximate counts within a window of the last w time steps with accuracy $\varepsilon n^{(i)}$. For each site locally, the data structure of Lee and Ting can track details of the last W arrivals (for a fixed parameter W) using $O(\varepsilon^{-1})$ space [16][3]. We begin the (active) window by instantiating such a data structure for $W = 2(\varepsilon/3)^{-1}$. After we have observed $n^{(i)} = 2^a$ items, we also instantiate a data structure for $W = 2^a(\varepsilon/3)^{-1}$ items, and run it for the remainder of the window: the omitted $n^{(i)} = O(\varepsilon W)$ items can be ignored without affecting the $O(\varepsilon W)$ error guarantee of the structure. Over the life of the window, $O(\log n^{(i)}/\varepsilon) = O(\log n^{(i)})$ (since $n^{(i)} > \varepsilon^{-1}$) such data structures will be instantiated. When the window is expiring, during the phases where $n^{(i)}$ is in the range $2^a(\varepsilon/3)^{-1} \ldots 2^{a-1}(\varepsilon/3)^{-1}$, the local site uses the instance of the data structure to monitor items and approximate counts, accurate to $\varepsilon n^{(i)}/3$. The structure allows the identification of a set of items which are frequent when this range begins (lines 4-5 in Algorithm 2). The structure also indicates how their approximate counts decrease as old instances of the items expire, and so when their (approximate) counts have decreased sufficiently, the coordinator can be updated (line 11). In this case, the estimated counts are monotone decreasing, so the communication cost does not alter. The space at each site is therefore dominated by the cost of these data structures, which is $O(\varepsilon^{-1} \log n^{(i)})$.

Theorem 2. *The above protocol for heavy hitters has a total communication cost of* $O(\frac{k}{\varepsilon} \log \frac{\varepsilon n}{k})$ *words per window. The space required at each site is* $O(\frac{1}{\varepsilon} \log n^{(i)})$*, and* $O(\frac{k}{\varepsilon})$ *at the coordinator to keep track of the current heavy hitters from each site.*

This protocol is optimal: it meets the communication lower bound for this problem stated in [4], and improves the upper bound therein. It similarly improves over the bound for the infinite window case in [20].

5 Quantiles

In this section, we study the problem of tracking the set of quantiles for a distribution in the sliding window model. Yi and Zhang [20] study this problem in the infinite window model, and provide a protocol with communication cost $O(\frac{k}{\varepsilon} \log n \log^2 \frac{1}{\varepsilon})$. As a byproduct, our solution slightly improves on this. The improvement over the best known solution for the sliding window model is more substantial.

In order to achieve small space and communication, we make use of the data structure of Arasu and Manku [2], referred to as the AM structure. The AM

[3] The λ-counter data structure defined therein can be extended to store timestamps in addition to items, which makes it sufficient for our purpose.

structure stores the ε-approximate quantiles over a sequence of W items, for W fixed in advance. The W items are divided along the time axis into blocks of size εW, and summaries are built of the blocks. Specifically, at level 0, an ε_0-approximate quantile summary (of size $1/\varepsilon_0$) is built for each block, for some ε_0 to be determined later. An ε-approximate quantile summary for a set of m items can be computed by simply storing every tth item in the sorted order, for $t = \varepsilon m$: from this, the absolute error in the rank of any item is at most t. Similarly, summaries are built for levels $\ell = 1, \ldots, \log(1/\varepsilon)$ with parameter ε_ℓ by successively pairing blocks in a binary tree: level ℓ groups the items into blocks of $2^\ell \varepsilon W$ items. Using this block structure, any time interval can be decomposed into $O(\log(1/\varepsilon))$ blocks, at most two from each level, plus two level-0 blocks at the boundaries that partially overlap with the interval. Ignoring these two boundary level-0 blocks introduces an error of $O(\varepsilon W)$. The blocks at level ℓ contribute an uncertainty in rank of $\varepsilon_\ell 2^\ell \varepsilon W$. Hence if we choose $\varepsilon_\ell = 1/(2^\ell \log(1/\varepsilon))$, the total error summed over all $L = \log(1/\varepsilon)$ levels is $O(\varepsilon W)$.

The total size of the structure is $\sum_\ell (1/\varepsilon_\ell \cdot 1/(2^\ell \varepsilon)) = O(1/\varepsilon \cdot \log^2(1/\varepsilon))$. Rather than explicitly storing all items in each block and sorting them to extract the summary, we can instead process the items in small space and a single pass using the GK algorithm [12] to summarize the active blocks. This algorithm needs space $O(1/\varepsilon_\ell \cdot \log \varepsilon^2 2^\ell W)$ for level ℓ. When a block completes, i.e., has received $2^\ell \varepsilon W$ items, we produce a compact quantile summary of size $1/\varepsilon_\ell$ for the block using the GK summary, and discard the GK summary. The space required is dominated by the GK algorithm at level $\ell = \log(1/\varepsilon)$, which is $O(1/\varepsilon \cdot \log(1/\varepsilon) \log(\varepsilon W))$.

The Forward Problem. Recall that in the forward problem, the coordinator needs to estimate ranks of items from site i with error at most $\varepsilon n^{(i)}$, where $n^{(i)}$ is the current number of items received since time t_0. To achieve this, we build multiple instances of the above structure for different values of W. When the $n^{(i)}$-th item arrives for which $n^{(i)}$ is a power of 2, the site starts building an AM structure with $W = n^{(i)}/\varepsilon$. Whenever a block from any level of any of the AM structures completes, the site sends it to the coordinator. This causes communication $O(1/\varepsilon \cdot \log^2(1/\varepsilon) \log n^{(i)})$ for the entire active window (the communication is slightly less than the space used, since only summaries of complete blocks are sent). After the $n^{(i)}$-th item has arrived, all AM structures with $W < n^{(i)}$ can be discarded.

We argue this is sufficient for the coordinator to track the quantiles of the active window at any time. Indeed, when the $n^{(i)}$-th item arrives, the site has already started building an AM structure with some W that is between $n^{(i)}$ and $2n^{(i)}$,[4] and the completed portion has been communicated to the coordinator. This structure gives us quantiles with error $O(\varepsilon W) = O(\varepsilon n^{(i)})$, as desired. Note that the site only started building the structure after εW items passed, but ignoring these contributes error at most εW.

[4] The special case $n^{(i)} \leq 1/\varepsilon$ is handled by simply recording the first $1/\varepsilon$ items exactly.

The Backward Problem. To solve the backward problem, we need a series of AM structures on the last W items of a fixed window so that we can extract quantiles when this fixed window becomes the expiring window. Fortunately the AM structure can be maintained easily so that it always tracks the last W items. Again for each level, new items are added to the latest (partial) block using the GK algorithm [12]; when all items of the oldest block are outside the last W, we remove the block.

For the current fixed window starting from t_0, we build a series of AM structures, as in the forward case. The difference is that after an AM structure is completed, we continue to slide it so as to track the last W items. This remains private to the site, so there is no communication until the current active window concludes. At this point, we have a collection of $O(\log(n^{(i)}))$ AM structures maintaining the last W items in the window for exponentially increasing W's. Then the site sends the summaries for windows of size between ε^{-1} and $2n^{(i)}$ to the coordinator, and the communication cost is the same as in the forward case. To maintain the quantiles at any time t, the coordinator finds the smallest AM structure (in terms of coverage of time) that covers the expiring window $[t-w, t_0)$, and queries that structure with the time interval $[t-w, t_0)$. This will give us quantiles with error $O(\varepsilon W) = O(\varepsilon C_i(t-w, t_0))$ since $W \leq 2C_i(t-w, t_0)$.

Theorem 3. *The above protocol for quantiles has a total communication cost of* $O(k/\varepsilon \log^2(1/\varepsilon) \log(n/k))$ *words per window. The space required at each site is* $O(\frac{1}{\varepsilon} \log^2(1/\varepsilon) \log(\varepsilon n^{(i)}))$*, and* $O(\frac{k}{\varepsilon} \log^2(1/\varepsilon) \log(\varepsilon n/k))$ *at the coordinator to keep copies of all the data structures.*

Yi and Zhang [20] show a lower bound of $\Omega(\frac{k}{\varepsilon} \log \frac{\varepsilon n}{k})$ messages of communication (for the infinite window case), so our communication cost is near-optimal up to polylogarithmic factors. Meanwhile, [4] provided an $O(k/\varepsilon^2 \cdot \log(n/k))$ cost solution, so our protocol represents an asymptotic improvement by a factor of $O(\frac{1}{\varepsilon \log^2(1/\varepsilon)})$. We leave it open to further improve this bound: removing at least one $\log(1/\varepsilon)$ term seems feasible but involved, and is beyond the scope of this paper.

6 Other Functions

The problems discussed so far have the nice property that we can separately consider the monitored function for each site, and use the additivity properties of the function to obtain the result for the overall function. We now discuss some more general functions that can also be monitored under the same model.

Distinct Counts. Given a universe of possible items, the distinct counts problem asks to find the number of items present within the sliding window (counting duplicated items only once). The summary data structure of Gibbons and Tirthapura can solve this problem for a stream of items under the infinite window semantics [10]. A hash function maps each item to a level, such that the probability of being mapped to level j is geometrically decreasing. The algorithm

tracks the set of items mapped to each level, until $O(1/\varepsilon^2)$ distinct items have been seen there, at which point the level is declared "full". Then the algorithm uses the number of distinct items present in the first non-full level to estimate the overall number of distinct items.

This leads to a simple solution for the active window: each site independently maintains an instance of the data structure for the window, and runs the algorithm. Each update to the data structure is echoed to the coordinator, ensuring that the coordinator has a copy of the structure. The communication cost is bounded by $O(1/\varepsilon^2 \log n^{(i)})$. The coordinator can merge these summaries together in the natural way (by retaining the set of items mapped to the same level from all sites) to get the summary of the union of all streams. This summary therefore accurately summarizes the distinct items across all sites.

The solution for the expiring window is similar. Each site retains for each level the $O(1/\varepsilon^2)$ most recent distinct arrivals that were mapped to that level, along with their timestamps. This information enables the distinct count for any suffix of the window to be approximated. This data structure can then be shared with the coordinator, who can again merge the data structures straightforwardly. The total communication required is $O(\frac{k}{\varepsilon^2} \log \frac{n}{k})$ over all k sites.

Entropy. The (empirical) entropy of a frequency distribution is $\sum_j \frac{f_j}{n} \log \frac{n}{f_j}$, where f_j denotes the frequency of the jth token. As more items arrive, the entropy does not necessarily vary in a monotone fashion. However, the amount by which it can vary is bounded based on the number of arriving items: specifically, for m new arrivals after n current arrivals, it can change by at most $\frac{m}{n} \log(2n)$ [1]. This leads to a simple protocol for the forward window case to track the entropy up to additive error ε: given n current items, each site waits to see $\frac{\varepsilon n}{\log(2n)}$ new arrivals, then communicates its current frequency distribution to the coordinator. Within a window of n_i arrivals, there are at most $O(\frac{1}{\varepsilon} \log^2 n_i)$ communications.

For the backward case, the protocol has to send the frequency distribution when the number of items remaining reaches various values. This can be arranged by use of the exponential histogram outlined in Section 3: for each timestamp stored, it keeps the frequency distribution associated with that timestamp. When a timestamp is "merged", and dropped, the corresponding distribution is dropped also. Thus, the space required is $O(\frac{1}{\varepsilon} \log(\varepsilon n^{(i)}))$ entries in the histogram. The histogram introduces some uncertainty into the number of items remaining, but after rescaling of parameters, this does not affect the correctness.

When the domain is large, the size of the frequency distributions which must be stored and sent may dominate the costs. In this case, we replace the exact distributions with compact *sketches* of size $\tilde{O}(\frac{1}{\varepsilon^2})$ [13]. The coordinator can combine the sketches from each site to obtain a sketch of the overall distribution, from which the entropy can be approximated.

Geometric Extents: Spread, Diameter and convex Hull. Given a set of points in one dimension p_i, their *spread* is given by $(\max_i p_i - \min_i p_i)$. The forward case is easy: send the first two points to the coordinator, then every time a new point causes the spread to increase by a $1 + \varepsilon/2$ factor, send the

new point to the coordinator. This ensures that spread is always maintained up to a $(1+\varepsilon)$ factor, and the communication is $O(\frac{1}{\varepsilon}\log R)$, where R is the ratio between the closest and furthest points in the input, a standard factor in computational geometry. For the backward case, we can use the algorithm of Chan and Sadjad [5] to build a summary of size $O(\frac{1}{\varepsilon}\log R)$ as the points arrive in the active window, and communicate this to the coordinator to use for the expiring window. Lastly, observe that spread of the union of all points can be approximated accurately from the union of points defining the (approximate) spread for each site.

The diameter of a point set in two (or higher) dimensions is the maximum spread of the point set when projected onto any line. A standard technique is to pick $O(1/\varepsilon^{(d-1)/2})$ uniformly spaced directions in d dimensions, and project each input point onto all of these directions: this preserves the diameter up to a $(1+\varepsilon)$ factor, since there is some line which is almost parallel to the line achieving the diameter. This immediately gives a protocol for diameter with cost $O(\frac{1}{\varepsilon}^{(d+1)/2}\log R)$, by running the above protocol for each of the $O(1/\varepsilon^{(d-1)/2})$ directions in parallel. A similar approach also maintains the approximate convex hull of the point set, by observing that the convex hull of the maximal points in each direction is approximately the convex hull of all points.

7 Concluding Remarks

The forward/backward framework allows a variety of functions to be monitored effectively within a sliding window, and improves over the results in prior work [4]. The underlying reason for the complexity of the the analysis of the protocols previously proposed is that they focus on the current count of items at each site. This count rises (due to new arrivals) and falls (due to expiry of old items). Capturing this behavior for heavy hitters and quantiles requires the analysis of many cases: when an item becomes frequent through arrivals; when an item becomes frequent because the local count has decreased; when an item becomes infrequent due to arrivals of other items; when an item becomes infrequent due to expiry; and so on. By separating streams into streams of only arrivals and streams of only expirations, we reduce the number of cases to consider, and remove any interactions between them. Instead, we just have to track the function for two different cases. This allowed a much cleaner treatment of this problem, and opens the door for similar analysis of other monitoring problems.

References

1. Arackaparambil, C., Brody, J., Chakrabarti, A.: Functional Monitoring without Monotonicity. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 95–106. Springer, Heidelberg (2009)
2. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: ACM Principles of Database Systems (2004)

3. Busch, C., Tirthapura, S., Xu, B.: Sketching asynchronous streams over sliding windows. In: ACM Conference on Principles of Distributed Computing (PODC) (2006)
4. Chan, H.-L., Lam, T.-W., Lee, L.-K., Ting, H.-F.: Continuous monitoring of distributed data streams over a time-based sliding window. In: Symposium on Theoretical Aspects of Computer Science, STACS (2010)
5. Chan, T.M., Sadjad, B.S.: Geometric Optimization Problems Over Sliding Windows. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 246–258. Springer, Heidelberg (2004)
6. Cormode, G.: Continuous distributed monitoring: A short survey. In: Algorithms and Models for Distributed Event Processing, AlMoDEP (2011)
7. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed, functional monitoring. In: ACM-SIAM Symposium on Discrete Algorithms (2008)
8. Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Optimal sampling from distributed streams. In: ACM Principles of Database Systems (2010)
9. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. In: ACM-SIAM Symposium on Discrete Algorithms (2002)
10. Gibbons, P., Tirthapura, S.: Estimating simple functions on the union of data streams. In: ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 281–290 (2001)
11. Gibbons, P., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: ACM Symposium on Parallel Algorithms and Architectures (SPAA) (2002)
12. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: ACM SIGMOD International Conference on Management of Data (2001)
13. Harvey, N.J.A., Nelson, J., Onak, K.: Sketching and streaming entropy via approximation theory. In: IEEE Conference on Foundations of Computer Science (2008)
14. Keralapura, R., Cormode, G., Ramamirtham, J.: Communication-efficient distributed monitoring of thresholded counts. In: ACM SIGMOD International Conference on Management of Data (2006)
15. Kuhn, F., Locher, T., Schmid, S.: Distributed computation of the mode. In: ACM Conference on Principles of Distributed Computing (PODC), pp. 15–24 (2008)
16. Lee, L., Ting, H.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: ACM Principles of Database Systems (2006)
17. Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams. In: International Conference on Database Theory (2005)
18. Patt-Shamir, B.: A note on efficient aggregate queries in sensor networks. In: ACM Conference on Principles of Distributed Computing (PODC), pp. 283–289 (2004)
19. Sharfman, I., Schuster, A., Keren, D.: A geometric approach to monitoring threshold functions over distributed data streams. In: ACM SIGMOD International Conference on Management of Data (2006)
20. Yi, K., Zhang, Q.: Optimal tracking of distributed heavy hitters and quantiles. In: ACM Principles of Database Systems, pp. 167–174 (2009)