Tracking Matrix Approximation over Distributed Sliding Windows

Haida Zhang Zengfeng Huang
School of CSE, UNSW
{haida.zhang, zengfeng.huang}@unsw.edu.au

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Wenjie Zhang Xuemin Lin
School of CSE, UNSW
{zhangw, lxue}@cse.unsw.edu.au

***Abstract*—In many modern applications, input data is represented as matrices and often arrives continuously. The ability to summarize and approximate data matrices in streaming fashion has become a common requirement in many emerging environments. In these applications, input data is usually generated at multiple distributed sites and simply centralizing all data is often infeasible. Therefore, novel algorithmic techniques are required. Furthermore, in most of these applications, queries must be answered solely based on the recently observed data points (e.g., data collected over the last hour/day/month), which makes the problem even more challenging. In this paper, we propose to study the problem of tracking matrix approximations over distributed sliding windows. In this problem, there are m distributed sites each observing a stream of d-dimensional data points. The goal is to continuously track a small matrix B as an approximation to A_w, the matrix consists of data points in the union of the streams which arrived during the last W time units. The quality of the approximation is measured by the *covariance error* $\|A_w^TA_w - B^TB\|/\|A\|_F^2$ [1], and the primary goal is to minimize communication, while providing provable error guarantee. We propose novel communication-efficient algorithms for this problem. Our sampling based algorithms continuously track a weighted sample of rows according to their squared norms, which generalize and simplify the sampling techniques in [2]. We also propose deterministic tracking algorithms that only require one-way communication and provide better error guarantee. All of our algorithms have provable guarantees, and extensive experimental studies on large real and synthetic datasets validate our theoretical claims and demonstrate the efficiency of these algorithms.**

I. Introduction

In many applications, including network traffic monitoring, financial data analysis, and real-time anomaly detection, data is received as continuous high-volume streams. Under such settings, decisions usually have to be made in real time (e.g., detection of DDoS attacks), and thus traditional database query processing methods are not suitable for such applications. The ability to summarizing massive streaming data continuously becomes crucial, i.e., dynamically maintaining a compact *summary* (a.k.a. synopsis or sketch) of the input stream, which can be used to provide *approximate* query answers.

In large-scale streaming processing applications, input data is usually *distributed*: streaming tuples are continuously received at multiple, possibly geographically dispersed, distributed sites (sensor nodes, routers, smart phones, etc). Efficiently tracking the value of a function over the union of streams has become a key procedure to support real-time decision making such as detection of fraudulent transactions, user interest prediction, and network health monitoring. This motivates the *distributed monitoring model* [3], which has attracted great attention in recent years (e.g. [4] [5] [2] [6] [7]). In this model, there are multiple distributed sites, each observing a disjoint stream of items, and a single coordinator, whose goal is to monitor (or track) a function over the union of the streams *at all times*. In such distributed applications, *communication efficiency* is always a primary concern either due to limited bandwidth or many other reasons. For example, in sensor networks, it is usually impractical to install new batteries for geographically dispersed sensor nodes, and thus power consumption is typically the main concern. As the batteries drain for sending messages exceeds by several orders of magnitude the drain for local operations within a node, reducing communication is critical to sensor networks [8] [9].

Despite the success of the distributed monitoring model, the bulk of prior works focus on simple aggregate queries (e.g., counts, item frequencies, and quantiles) and low-dimensional data. In many tasks of machine learning, information retrieval, and large-scale data analytics, the input data is represented as large matrices [10] [11] [12] [13]. For example, in textual analysis, the set of documents is modeled as a matrix whose rows correspond to different documents and columns correspond to words; in image analysis, each image is represented as a row of a matrix containing either pixel values or other derived feature values. Such data is often huge, complex, and continuously generated at multiple sources, so summarizing and approximating such large matrix data has become a common requirement in many emerging application environments. Recently a new line of research, represented by [12], focuses on the *row-update streaming model*, in which, each item in the stream is a row of a matrix; the goal is to maintain a good approximation to the matrix consisting of all the rows received so far. We have seen a flurry of activities in the area of matrix approximation on centralized streams [14] [15] [16] [17]. However, little is known on distributed tracking of matrix approximations until very recently [1].

In streaming processing, each item observed naturally carries a timestamp, and capturing recent data is usually more interesting. For instance, Google Trends tracks a list of popular search-terms within the past 24 hours; in network analysis, to detect DDoS attacks, one should track frequency statistics over a short period rather than the entire history [18]. The *sliding window model* is arguably one of the most prominent and intuitive models for handling this time decaying issue [19] [20] [21], which considers only a window of the most recent items seen in the stream thus far. In recent years, distributed monitoring over sliding windows has also been widely studied for simple aggregate queries (e.g. [22] [2] [23] [24]).

Motivations. In this paper, we study the problem of continuously tracking a matrix approximation over distributed sliding windows. More precisely, we propose communication-efficient, space-efficient and processing-time-efficient algorithms for maintaining a *covariance sketch* over sliding windows in the distributed monitoring model. Our problem is motivated by many applications in sensor networks, distributed databases, cloud computing, etc, where data are collected over a network with rapid speed, and queries must be answered continuously, based on the data collected over a recent time period. We provide two concrete applications for our matrix approximation techniques.

- **Approximate PCA for Change Detection.** In *principle component analysis*, the goal is to compute an orthonormal basis of a lower dimensional subspace which captures the variance of the data as much as possible. Interestingly, it was shown that the top-k right singular vectors of a *covariance sketch* approximate the optimal PCA basis of the original matrix well in a certain sense [14]. Note that a covariance sketch is a type of matrix approximation, which will be formally defined in the next section. Therefore, we can perform PCA on a much smaller sketch matrix instead of on the original one. A particular application of PCA is to detect changes in multidimensional data streams [25]. More concretely, changes are detected by comparing the PCA basis of the *testing window* (i.e., the current window) to the PCA basis of a fixed *reference window* which has been extracted earlier. Hence, the ability to continuously maintain an (approximate) PCA basis of the current window is crucial to this approach. This paper provides a way to continuously track such a basis in distributed environments, where data is generated at multiple distributed nodes. Under the setting of *online* PCA, covariance sketch is also served as the main building block to improving the efficiency [26] [27].
- **Anomaly detection.** In a recent work [15], covariance sketch is used as the main tool for streaming anomaly detection in the centralized setting. Given a set of non-anomalous data points observed so far (represented as a matrix A), function $f(A, x)$ is used to measure the anomaly score of a new data point x. However it is infeasible to compute $f(A, x)$ exactly. Then it is shown that if B has small covariance error w.r.t. A, then $f(A, x)$ can be approximated accurately by $f(B, x)$, which can be computed much more efficiently in terms of running time and space. Note that in some situations, anomaly is supposed to be detected based on a sliding window of input (due to *concept drift*), and also input data might be generated at multiple distributed sites. [15] did not address these issues, and we believe, our algorithms can be used as tools for window-based anomaly detection over distributed streams.

A. Problem definitions

Before giving formal definitions for our problem, we first provide some linear algebra and other notations we will use.

Notation	Descriptions
a_j	a row in the matrix stream
d	the dimension of a row
R	the maximum ratio of squared norms of rows
W	window size
A_w	the matrix within the current sliding window
N	the maximum number of rows in a sliding window
m	the number of sites
ℓ	the size of sample set
B	a matrix approximation to A_w
ε	error parameter

Table I: Symbols and descriptions

Notations. We always use m for the number of sites, and d for the dimension of each row. R is the maximum ratio of squared norms of any two rows in the matrix and N is the maximum number of rows in a sliding window. We remark that our protocols do not need to know R or N in advance. A_w refers to the matrix that consists of all *active* rows, i.e., rows included by current window. For a d-dimensional vector x, $\|x\|$ is the ℓ_2 norm of x. We use x_i to denote the ith entry of x. Let $A \in \mathbb{R}^{n\times d}$ be a matrix of dimension $n \times d$ with $n > d$. We use a_i to denote the ith row of A, and $a_{i,j}$ for the (i, j)-th entry of A. We write the (reduced) singular value decomposition of A as $(U, \Sigma, V) = \mathsf{SVD}(A)$. The computation time of standard SVD algorithms is $O(\min(nd^2, n^2d))$. We use $\|A\|_2$ or $\|A\|$ to denote the spectral norm of A, which is the largest singular value of A, and $\|A\|_F$ for the *Frobenius Norm*, which is $\sqrt{\sum_{i,j} a_{i,j}^2}$. Given another matrix B with the same number of columns as A, $[A; B]$ is the matrix formed by concatenating the rows of A and B. See Table I for a summary.

Covariance sketch. In this paper, we measure the quality of matrix approximation by the well-known covariance error. More formally, given a target $n \times d$ matrix A and an approximation parameter ε, we call another matrix $B \in \mathbb{R}^{\ell\times d}$ an ε-covariance sketch of A, if

$$\|A^TA - B^TB\|/\|A\|_F^2 \leq \varepsilon.$$

In practice, we want $\ell \ll n$. We will call

$$\textbf{cova-err}(A, B) = \|A^TA - B^TB\|/\|A\|_F^2$$

the covariance error of B with respect to A. Alternatively, it is known [12] that the covariance error can be equivalently defined as

$$\max_{x:\|x\|=1} |\|Ax\|^2 - \|Bx\|^2|/\|A\|_F^2.$$

Intuitively, a covariance sketch preserves the norm (or length) of A in any direction x.

Tracking covariance sketch over distributed time-based sliding windows. Formally, we model a data stream as a infinite sequence $S = \{(a_i, t_i) \mid i = 1, \cdots, \infty\}$, where a_i is a row (or record) with d numerical attributes and t_i is the timestamp of a_i. Assume t_{now} is the current time, given a window size W, we use A_w to denote the matrix formed by all the rows whose timestamps are in the range $(t_{now} - W, t_{now}]$, which is our target matrix. We call a row *active*, if it is in A_w. Note that the number of rows in a time window could vary drastically over time. If the window size W is infinite, the model degenerates to the standard distributed monitoring model, and we will simply call it infinite-window model.

At time t_i, the record a_i appears at exactly one of the m distributed sites $S_1, S_2, \cdots, S_m$. The goal is to maintain or track a smaller matrix B which has covariance error at most ε to A_w at the coordinator site C. Each of the m sites only communicates with the coordinator C through a two-way communication channel. Notice that models allow arbitrary point-to-point communication can be simulated by this model within a multiplicative factor of 2 in communication cost.

We remark that there is an alternative sliding window model called *sequence-based sliding window*, which considers the last W items in the stream. However, as argued in [2], time-based model is often more useful in practice. Besides, due to subtle model issues, sequence-based model is quite different from time-based sliding window in the distributed setting (although it is a special case of time-based model in the centralized setting), thus requires a different set of techniques. Therefore, attention is focused on time-based sliding window in this paper. See [2] for more discussions on models.

B. Our contributions

In this paper, we provide several algorithms/protocols for our problem based on different techniques. The protocols can be divided into two categories: (1) *sampling based*, and (2) *deterministic tracking*.

Sampling based. In sampling based protocols, we aim to pick random samples of rows in the sliding window with probability proportional to their squared norms. Random sampling is a fundamental tool for analyzing big data, which has been extensively studied in the centralized setting [28]. However, it is highly non-trivial to extend it to the distributed setting. The problem in the distributed sliding window model was only recently studied in [2], but so far all the techniques only work for unweighted sampling, i.e., each item is sampled with the same probability. In this paper, we provide first protocols for *continuous weighted sampling over distributed sliding windows*.

In the centralized setting, there are two well-known methods for weighted sampling, namely *priority sampling* of [29] and *ES sampling* of [30]. In this paper, we propose a general framework, which extends both methods to the distributed sliding window model. Our framework works for both sampling with and without replacement, denoted as SWR and SWOR respectively.

Even for the special case of unweighted sampling, our protocols not only achieve the same optimal guarantees on communication cost and space usage as in [2], but are arguably more intuitive and much easier to implement. In particular, the unweighted sampling protocol in [2] maintains a complicated distributed data structure called *level-sampling data structure*, while ours do not need any of such structures except standard priority queues to improve processing time.

Deterministic tracking with one-way communication. We also provide two deterministic tracking algorithms based on two algorithmic frameworks. One simple but crucial observation is that, to track a global covariance sketch, the coordinator only needs to track each of the m local sliding-window matrices separately, i.e., at any time, the coordinator maintains a ε-covariance sketch for each site separately.

To do so, the first framework uses a surprisingly simple idea. Roughly speaking, each site tracks the covariance error between its local sliding-window matrix and the sketch that coordinator currently has, and will send a message only when the error exceeds a threshold. This simple idea is actually quite general, and also works for simple aggregate queries such as counting, item frequencies, and order statistics, which can simplify the algorithms in [24]. We also provide an alternative way to track each local matrix. In this second approach, we combined the *forward-backward* framework of [24] with *iterative matrix sketching* [12].

One property of our deterministic protocols is they only use one-way communication, i.e., all communication is from sites to the coordinator, is desirable in many applications.

Theoretical analysis and experimental studies. We provide rigorous analysis for each of our protocols. The asymptotic bounds on communication cost and space usage are listed in Table II. Moreover, we conduct extensive experimental studies on both real and synthetic datasets to test and compare the performances of proposed algorithms in practice.

C. Related work

Matrix approximation problems have been extensively studied in the centralized streaming model. The row-update streaming model is most related to our study. In this model, each item in the stream is a row of a matrix; the goal is to maintain a good approximation to the matrix consisting of all the rows received so far with limited space. Liberty [12] proposed the Frequent Direction (FD) algorithm, which maintains a ε-covariance sketch using $O(d/\varepsilon)$ space. [14] [16] improved and generalized the original FD. More recently, [17] provided algorithms for maintaining covariance sketch in the sliding window model. But none of their results can be applied in distributed setting without incurring high communication cost. Other error measures of matrix approximation are also widely studied, we refer to a recent survey by Woodruff [31].

In the popular distributed monitoring model, numerous work has been done for tracking simple aggregate queries [4] [5] [2] [6] [7]. See the recent survey by Cormode [3] for more references in this area. A work closely related to ours is [1], which considers the problem of tracking covariance sketch in the standard distributed monitoring model. But none of their techniques works if a sliding window of input is considered.

Tracking simple aggregate queries, such as counts, heavy hitter, and quantiles, in the distributed sliding window model has also attracted lots of attention in recent years [22] [2] [24] [23]. But to the best of our knowledge, there is no comprehensive study of tracking matrix approximation in this model. Cormode et. al. [2] considered the problem of tracking random samples over distributed sliding windows, but current techniques only works for unweighted sampling, and thus cannot work for tracking matrix approximation.

Outline. Section II presents sampling algorithms for distributed time-based sliding window, and explores some optimization strategies to reduce communication cost with theoretical guarantees. Section III introduces deterministic algorithms

Algorithms	Communication (Words)	Space per site (Words)
Priority Sampling	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR) + m\log(NR)$	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR)$
ES Sampling	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR) + \frac{m}{\varepsilon}\log(NR)$	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR)$
DA1	$\frac{md}{\varepsilon}\log(NR)$	$d^2 + \frac{d}{\varepsilon^2}\log(NR)$
DA2	$\frac{md}{\varepsilon}\log(NR)$	$\frac{d}{\varepsilon^2}\log(NR)$

Table II: Asymptotic bounds on communication and space of our protocols for tracking covariance sketch over distributed sliding windows. The cost of sampling protocols holds with high probability.

to tracking sum and tracking matrix over distributed time-based sliding window, and provides firm proofs for communication complexity and space complexity. Section IV evaluates all the introduced algorithms using extensive experiments. Finally we conclude the paper in Section V.

II. Sampling algorithms

In this section, we describe our sampling based algorithms. Our goal is to maintain a random sample S of rows on the coordinator, where the rows are chosen with probabilities proportional to their squared norms. To produce a ε-covariance sketch with high probability, the sample size should be at least $\ell = O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$. In the centralized setting, there are two well-known techniques for random sampling on weighted items, namely ES sampling [30] and priority sampling [29]. Both of them use the following framework: to sample ℓ items without replacement, the algorithm assigns each item a random priority based on its weight, and picks the items with top-ℓ priorities as the sample set. The main difference between ES sampling and priority sampling is the way to assign priorities and create estimates.

We propose a general framework in the distributed sliding window model, which efficiently tracks the set of items with top-ℓ priorities regardless how their priorities are calculated, and thus works for both techniques mentioned above. The key challenge is that, when later arrivals are dominated by earlier items (in terms of priority), we cannot simply discard them. Otherwise, when earlier items expire, there may be insufficient later items kept to provide a sample of size ℓ. This challenge also exists in unweighted sampling (due to non-uniform arrival rate of items), and was resolved in [2] either by storing all active items or using a complicated distributed data structure. The weighted version is more challenging, but we provide a solution, which is more general (handling weights), uses the same communication and space, but is surprisingly simple.

Next, we will mainly focus on priority sampling. We first show how to do sampling without replacement, then discuss sampling with replacement. ES sampling can be applied similarly. For simplicity, we assume that there are always more than ℓ active rows. Otherwise, it will be clear that the coordinator retrieves all active rows in our protocols.

A. Priority Sampling without Replacement (PWOR)

In priority sampling, each item a_i with weight w_i is assigned a priority value $\rho_i = \frac{w_i}{u_i}$, where u_i is a random number uniformly chosen from $(0, 1)$. The goal is to track the set of items with top-ℓ priority values. And in the row sampling problem, each item is a row a_i with weight $w_i = \|a_i\|^2$.

A simple protocol. Our first protocol works as follows. At any time, all the sites maintain the same threshold τ, which is the ℓ-th largest priority value among all active rows. When a new row a_i is observed by the site S_j, it is assigned a priority value ρ_i defined as above. If ρ_i is greater than the threshold τ, the site sends (a_i, t_i, ρ_i) to the coordinator and discards it. Otherwise, if $\rho_i \le \tau$, the site stores it locally until there are ℓ rows in the same site that arrive later than a_i but have higher priority values than ρ_i or until a_i is sent to the coordinator[1].

The coordinator maintains a queue S, which contains the set of rows with top-ℓ priority values: (1) On receiving a new tuple (a_i, t_i, ρ_i) from a site, the coordinator inserts this tuple to S; updates τ properly and broadcasts the new τ to all sites. The row in S with the smallest priority value is moved to S' (candidate set). (2) When a row in S expires, the coordinator broadcasts a message to all sites; each site sends back the highest priority value among all local active rows; the coordinator finds the global highest value among m sites and the candidate set S', then retrieves the corresponding row and inserts it into the queue.

Our first protocol is presented in **Algorithm** 1. Each observed row a_i is assigned a priority value ρ_i (line 2). The tuple (a_i, t_i, ρ_i) is sent to coordinator if ρ_i is greater than the threshold τ (line 3,4), otherwise it is appended to a local queue Q (line 5). Then site j removes all rows that expire (line 7) and rows that are dominated by ℓ later rows with respect to priorities (line 8-11). On receiving samples, the coordinator appends them to the sample set S (line 13,14). Then coordinator removes all rows in S and S' that expire (line 15,16). Then we introduce how to update threshold. While the size of sample set is larger than ℓ, the coordinator moves the sample with the minimum priority value to candidate set S' (line 18-20), because such rows still have the possibility of becoming top-ℓ. While the size of sample set is smaller than ℓ (due to expiration), coordinator always requests among S' and m sites the row with the highest priority and adds it to sample set (line 21-23). Finally, the coordinator updates τ and broadcasts it to m sites if τ has changed (line 24-25).

We will need the definition of *ℓ-dominance*.

Definition 1. Given a sequence of rows $a_1, a_2, \cdots, a_N$, each of which is assigned a (random) priority value as in priority sampling. We say a_i is left ℓ-dominated if there are ℓ rows before a_i which have greater priority values than ρ_i. The right ℓ-dominance is defined analogously.

The correctness of the above algorithm is based on the following simple observation: if a_i is right ℓ-dominated by later rows, then a_i will never be in the top-ℓ before it expires. In our protocol each site only discards rows that have been right ℓ-dominated by later rows, and thus no potential top-ℓ row is discarded. Thereby, the correctness follows, since the coordinator always maintains the global top-ℓ rows among all rows currently stored in the whole system.

Despite the apparent simplicity, it is not clear whether this protocol has any theoretical guarantee on communication cost and space usage. We will give a formal analysis next.

[1]We remark that this is the key step to save space, which we will show achieves the same guarantee as the level-sampling structure in [2], but is much simpler.

Algorithm 1 Priority Sampling without replacement (PWOR)

```
 1: procedure ROWS_PROCESS(row $a_i$)                    ▷ at site $j$
 2:     Choose $u_i$ ∈Unif$(0, 1)$ and set $\rho_i = \frac{\|a_i\|^2}{u_i}$.
 3:     if $\rho_i \geq \tau$ then
 4:         Send $(a_i, \rho_i, t_i)$ to coordinator.
 5:     else Append $(a_i, \rho_i, t_i, count_i = 0)$ to the end of $Q$.
 6:     for $(a_p, \rho_p, t_p, count_p) \in Q$ do
 7:         if $t_p < t - W$ then Remove $(a_p, \rho_p, t_p, count_p)$.
 8:         if $\rho_i \geq \rho_p$ then
 9:             $count_p += 1$.
10:             if $count_p \geq \ell$ then
11:                 Remove $(a_p, \rho_p, t_p, count_p)$ from $Q$.
12: procedure ROW_PROCESS(time $t$)                ▷ at coordinator
13:     for $(a_i, \rho_i, t_i) = receive\_rows(t)$ do
14:         Insert $(a_i, \rho_i, t_i)$ to $S$.
15:     for $(a_p, \rho_p, t_p) \in S, S'$ do
16:         if $t_p < t - W$ then Remove $(a_p, \rho_p, t_p)$.
17: procedure THRESHOLD_UPDATE()                   ▷ at coordinator
18:     while $|S| > \ell$ do
19:         Find the minimum $\rho_p$ in $S$;
20:         Move $(a_p, \rho_p, t_p)$ to candidate set $S'$.
21:     while $|S| < \ell$ do
22:         Request highest $\rho_i$ among $S'$ and $m$ sites
23:         Retrieve $(a_i, \rho_i, t_i)$ and insert it to $S$
24:     Update $\tau$ to be the minimum priority in $S$
25:     Broadcast $\tau$ if $\tau$ has changed from last time
```

The next two lemmas are crucial to our analysis.

Lemma 1. *For a sequence of rows $\{a_1, \cdots a_N\}$ each assigned with a priority value ρ_i as in priority sampling. Given any τ, let $B_1 = \{a_i \mid \rho_i \geq \tau\}$ and $B_0 = \{a_i \mid \rho_i \geq \tau/2\}$. Then,*

$$\Pr[|B_0| \geq 4|B_1|] \leq e^{-\Omega(|B_0|)}.$$

The above lemma roughly says that, when there are exactly ℓ rows with priority values greater than τ, then, with high probability, the number of rows with priority values greater than $\tau/2$ cannot be too large. This is intuitively correct, since conditioned on the priority value is greater than $\tau/2$, it is greater than τ with good probability.

Proof: For any a_i, it was shown in [1] $\Pr[\rho_i \geq \tau|\rho_i \geq \tau/2] \geq 1/2$. For completeness, we include a proof here.

$$\begin{aligned}\Pr[\rho_i \geq \tau|\rho_i \geq \tau/2] &= \frac{\Pr[\rho_i \geq \tau, \rho_i \geq \tau/2]}{\Pr[\rho_i \geq \tau/2]}\\ &= \frac{\Pr[\rho_i \geq \tau]}{\Pr[\rho_i \geq \tau/2]}\\ &= \frac{\Pr[w_i/u_i \geq \tau]}{\Pr[w_i/u_i \geq \tau/2]}\\ &= \frac{\min(1, w_i/\tau)}{\min(1, 2w_i/\tau)} \geq 1/2\end{aligned}$$

Thus, for each row a_i in B_0, we have $\Pr[a_i \in B_1] \geq 1/2$. By a standard use of Chernoff bound, we have

$$\Pr[|B_1| \leq |B_0|/4] \leq e^{-|B_0|},$$

which proves the lemma. ■

Lemma 2. *For a sequence of N rows with the maximum ratio of squared norms bounded by R, the number of rows that are not left ℓ-dominated is bounded by $O(\ell \log \frac{NR}{\ell})$ with probability $1 - e^{-\Omega(\ell)}$. Similarly, the number of rows that are not right ℓ-dominated is also bounded by $O(\ell \log \frac{NR}{\ell})$ with probability $1 - e^{-\Omega(\ell)}$ by symmetry.*

Proof: At first, we will show a construction of another set that contains all qualified rows in Lemma 1. We maintain q buckets from B_1 to B_q. Each bucket B_j is assigned with a threshold $\tau_j = 2^{j-1}$, i.e., the minimum threshold is $\tau_1 = 1$ and the maximum threshold is $\tau_q = \frac{2NR}{\ell}$. On arrival of each row a_i, we assign it with a priority value $\rho_i = \|a_i\|^2/u_i$, where u_i is a random number uniformly drawn from $(0, 1)$. For j from 1 to q, if $\rho_i \geq \tau_j$, then a_i is added to B_j. Notice that each row can be added to multiple buckets, and every row is inherently qualified for B_1. However, when there have been ℓ rows in the union of $B_{j+1}, \cdots, B_q$, we close the entrance to B_j. In this manner, a row is not added to any bucket only when it is left ℓ-dominated. Therefore, the union of q buckets contains all rows in the sequence that are not ℓ-dominated.

The probability of a_i entering B_q is

$$\Pr(a_i \in B_q) = Pr(\frac{\|a_i\|^2}{u_i} \geq \frac{2NR}{\ell}) \leq \frac{\ell}{2N}.$$

By Chernoff bound, we have

$$\Pr(|B_q| > \ell) < e^{-\ell/3}.$$

From the proof of Lemma 1, we have

$$\Pr[\rho_i \geq \tau_{j+1}|\rho_i \geq \tau_j] \geq 1/2.$$

Let $Q_j = b_1, b_2, \cdots, b_t$ be the rows with priority no less than τ_j. By the above inequality, the probability that b_i is in Q_{j+1} is at least $1/2$. The x_i be the random variable attaining value 1 if $b_i \in Q_{j+1}$, and otherwise 0, and thus $\mathsf{E}[x_i] \geq 1/2$. Since the size of B_j is the first index T such that $\sum_{i=1}^{T} x_i = \ell$, T is a stopping time for the sequence $x_1, \cdots, x_t$. Then by Wald's equation (see e.g. [32]), it holds that

$$\mathsf{E}[\sum_{i=1}^{T} x_i] = \mathsf{E}[T] \cdot E[x_i] \geq \mathsf{E}[T]/2.$$

Since by definition, $\mathsf{E}[\sum_{i=1}^{T} x_i] = \ell$, it follows that $\mathsf{E}[T] \leq 2\ell$. The high probability result follows from a standard martingale argument, which we omit here.

We have proved that $|B_j| = O(\ell)$ for all j, and thus the total number of rows in all buckets is $O(\ell \log(NR))$. The case for right dominance is the same. ■

Lemma 3. *During a time window, the total communication cost is $O(d\ell \log(NR) + m\ell \log(NR))$, where N is the maximum number of rows in a window. Each site stores $O(\ell \log(NR))$ rows.*

Proof: We consider any fixed time interval $(t, t+W]$, and bound the number of rows sent in this time. Let $a_1, \cdots, a_N$ denote the sequence of rows arrive during $(t, t+W]$.

Consider the rows sent immediately when they are received. By definition, it is clear that the rows sent immediately are not left ℓ-dominated in the sequence $a_1, \cdots, a_N$. According to Lemma 2, the expected number of such rows is bounded by $O(\ell \log(NR))$ (the communication cost is $O(d\ell \log(NR))$ words). After a row is sent to the coordinator, the threshold needs to be updated, which incurs $O(m)$ communication, and thus $O(m\ell \log(NR))$ in total.

Secondly, when a row in S expires, at most one row will be sent to the coordinator. Note that all rows in S that expire during $(t, t+W]$ arrived in the time interval $(t-W, t]$, hence, it is sufficient to bound the number of rows sent to coordinator whose arrival time is between $t-W$ and t. However, as shown above, the total number of rows sent per window is $O(\ell \log(NR))$. Therefore, the number of rows sent due to expiration of earlier samples is at most $O(\ell \log(NR))$ per window. When a row in S expires, $O(m)$ communication is needed to locate the row with highest priority among all unsent rows (thus $O(m\ell \log(NR))$ words in total).

We next bound the space usage at any fixed time t'. Let $\Delta = b_1, \cdots, b_M$ be the sequence of rows observed in time interval $(t'-W, t']$ at site S_j. At time t', site S_j only maintains all rows that are not right ℓ-dominated in the sequence Δ. By Lemma 2, the number of rows stored by site S_j is bounded by $O(\ell \log(NR))$. ■

Drawback of Algorithm 1. In the above protocol, the threshold changes $O(\ell \log(NR))$ times per window. Each time it incurs additional $O(m)$ communication, leading to a $O(m\ell \log(NR))$ term in overall communication cost, which is significant when d is small. Moreover, every time τ changes, the whole system needs to be synchronized. In practice, frequent synchronizations greatly downgrade the overall performance of distributed systems. Motivated by these, we propose a lazy-broadcast protocol that significantly reduces the number of updates in τ.

An improved protocol: Lazy-broadcast protocol. In lazy-broadcast protocol, the number of samples maintained by the coordinator is between ℓ and 4ℓ, i.e. $\ell \le |S| \le 4\ell$, rather than exactly ℓ. The coordinator only changes the threshold when the condition is violated. More precisely, when the size of S exceeds 4ℓ, the coordinator increases τ to remove half of rows in S; when the size of S becomes ℓ, the coordinator halves τ to collect more rows from sites. Accordingly, each site always maintains rows that are not right 4ℓ-dominated.

Lazy-broadcast protocol is demonstrated in **Algorithm** 2. The procedures of processing new observed rows are the same as in **Algorithm** 1, thus omitted. After updating sample set S, if the coordinator finds the size of S exceeds 4ℓ, it assigns τ the 2ℓ-th priority value of all rows in sample set, and broadcasts it to all sites (line 3,4). The samples whose priority values are less than τ are moved to a candidate set S' (line 5,6), since each of them still has the chance to become top-ℓ in term of priority when τ decreases. If the coordinator finds the size of sample set S less than ℓ (due to expiration), it keeps halving the threshold, collecting valid rows from all sites and from S', and inserting such rows into S, until the size of S exceeds 2ℓ (line 7-12). On acquiring a new threshold τ from the coordinator, site S_j compares it with the old τ_j. If $\tau < \tau_j$, site S_j should send all active rows with priority values greater than τ to the

Algorithm 2 Lazy-broadcast protocol

```
1: procedure THRESHOLD_UPDATE()    ▷ at coordinator
2:     if $|S| \ge 4\ell$ then
3:         Assign $\tau$ the $2\ell$-th priority value in $S$.
4:         Broadcast $\tau$ to $m$ sites.
5:         for each $(a_p, \rho_p, t_p)$ in $S$ with $\rho_p < \tau$ do
6:             Move $(a_p, \rho_p, t_p)$ to $S'$.
7:     if $|S| \le \ell$ then
8:         while $|S| \le 2\ell$ do
9:             $\tau = \tau/2$;
10:            Broadcast $\tau$ to $m$ sites;
11:            Collect rows from $S'$ and all sites with $\rho_p > \tau$;
12:            Insert these rows into $S$.
13: procedure THRESHOLD_UPDATE(threshold $\tau$) ▷ at site $j$
14:     if $\tau < \tau_j$ then
15:         for $(a_p, \rho_p, t_p, count_p) \in S$ and $\rho_p \ge \tau_i$ do
16:             Send message $(a_p, \rho_p)$.
17:             Remove $(a_p, \rho_p, t_p, count_p)$.
18:     Set $\tau_j = \tau$.
```

coordinator (line 14-17). Finally, site j sets $\tau_j = \tau$ (line 18).

Analysis. The correctness of the protocol is easy to prove, which we omit. The space usage is the same. We next analyze the communication cost and the number of threshold updates.

Lemma 4. *During any time interval $(t, t+W]$, the communication cost of the lazy-broadcast protocol is $O(d\ell \log(NR) + m \log(NR))$, and the number of updates of τ is $O(\log(NR))$.*

Proof: Let us consider any fixed time t'. Assume τ is the threshold at time t', and we claim that there are at most 8ℓ active rows that have higher priority values than τ for any t' with high probability. Given this claim, we can prove the communication bound using the same argument as in the proof of Lemma 3. We next prove the above claim.

When τ does not change at t', the claim is trivially true. When τ increases at t', then by definition, there are only 2ℓ active rows that have higher priority values than τ. So the only case we need to consider is when τ was just halved. In this case, there might be many active rows stored in sites with higher priorities. However, with high probability, this cannot happen. To see that, we know τ is halved to $\tau/2$ only when there are less than 2ℓ active rows with priority values higher than τ. Conditioned on this, the number of active rows having priority values larger than $\tau/2$ is bounded by 8ℓ with high probability according to Lemma 1, and thus the claim follows.

In the lazy-broadcast protocol, for each sampled row, at the time when it is sent to the coordinator, its priority values rank into top-4ℓ in the sliding window. Thus the set of rows transferred in a lazy-broadcast protocol is a subset of the samples transferred while the coordinator is maintaining the exact top-4ℓ rows, whose size is expected $O(\ell \log(NR))$.

Now we bound the number of broadcasts, i.e., the number of times τ is updated. Consider two consecutive increases of τ at t_1 and t_2 respectively. At time $t_1 + 1$, the size of S becomes 2ℓ, while at t_2 the size of S is 4ℓ. Note that we cannot simply conclude that the exact 2ℓ rows are sent from all sites, since the rows in S' could be added back to S. We consider two cases.

If τ does not decrease between t_1 and t_2, then no rows in S' will be added back to S. Therefore, at least 2ℓ rows are sent from sites during this time period. Otherwise, if τ decreases at some time $t^* \in (t_1, t_2)$, then there are at least ℓ rows in S from time t_1 have expired. As a result, between any two consecutive increases of τ, either 2ℓ rows are sent to the coordinator or at least ℓ samples expire. During a time window, the total number of rows sent is $O(\ell \log(NR))$ and the number of sampled rows get expired is at most $O(\ell \log(NR))$, and thus the number of times that τ increases is at most $O(\log(NR))$.

Now consider the moment when $|S|$ becomes ℓ, i.e., τ needs to be halved. Assume β is the minimum weight among all rows, and thus the maximum weight is $R\beta$. We claim that, with high probability, $\tau \leq NR\beta$. To see that, we have

$$\Pr[\rho_i \geq NR\beta] \leq 1/N.$$

Given any set of rows of size ℓ, the probability that the minimum priority among then is higher than $NR\beta$ is at most $1/N^\ell$. Via union bound, the probability that the ℓ-th largest priority value in a set of N rows is higher than $NR\beta$ is at most

$$\binom{N}{\ell} \cdot \frac{1}{N^\ell} \leq e^{-\ell}.$$

On the other hand, the minimum possible priority value is β, and thus the number of iterations of the while loop in line 8 of Algorithm 2 is at most $O(\log(NR))$. After this, $|S| \geq 2\ell$, which means before the next time when $|S| = \ell$, there are at least ℓ rows in S expiring. Therefore, the number of times that $|S|$ reaches ℓ is $O(\log(NR))$ during a time window (since at most $O(\ell \log(NR))$ sampled rows will expire during a time window as shown in the proof of Lemma 3). Since we have shown, each time $|S|$ becomes ℓ, it incurs $O(\log(NR))$ updates of threshold, the threshold τ is updated for at most $O(\log^2(NR))$ times during a time window. This can be reduced to $O(\log(NR))$ by a more careful analysis. We omit full details for space reasons. ∎

Output a covariance sketch. It was known that (e.g. [1]), we only need to sample $\ell = O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ rows to achieve ε-covariance error. To get a valid covariance sketch, we rescale all sampled rows properly to get unbiased estimators. For priority sampling, we assign a sampled row a_i with a new weight $v_i = \max\{\|a_i\|^2, \tau_\ell\}$ [29], where τ_ℓ is the ℓ-th largest priority value, i.e., set $a_i = \frac{v_i}{\|a_i\|^2} a_i$ for all sampled rows. By stacking all rescaled rows together, the matrix $B = [a_1; a_2; \cdots; a_\ell]$ is an ε-covariance sketch.

Priority Sampling with Replacement (PWR). We show a similar result on priority sampling with replacement. This can be done by maintaining ℓ independent samplers in the coordinator, each of which uses the above PWOR protocol to maintain $O(1)$ samples.

Although the communication cost is similar to PWOR, this PWR protocol needs to maintain ℓ independent thresholds, which requires $O(\ell \log(NR))$ threshold synchronizations. To avoid the large cost of broadcasting thresholds, we can use a similar sharing-threshold method as in [2] to solve our problem.

Sharing-threshold protocol. The sharing-threshold protocol uses the same idea of updating threshold as that of lazy-broadcast protocol, i.e., we usually have more qualified samples than ℓ, thus make the threshold updates less frequent. We use 4ℓ samplers in the sites and the coordinator, $Sp_1, Sp_2, \cdots, Sp_{4\ell}$, each of which maintains $O(1)$ samples. By guaranteeing there are at least ℓ qualified samplers, i.e., each sampler holds at least 2 samples, we have enough samples to create a good estimate based on with-replacement scheme. The reason why each qualified sampler should have 2 samples is that, a sampler in priority sampling [29] always needs top $k+1$ samples in terms of priorities to gain a sample set of size k, since the $(k+1)$-th priority value ρ_{k+1} is only used for rescaling samples and the corresponding row will be excluded from sample set.

Row update. All samplers share the same threshold τ, which is also initialized with 1 at the beginning. Whenever a row a_i is observed, site S_j generates 4ℓ independent random variables, i.e. u_i^k drawn uniformly from $(0,1)$ for $1 \leq k \leq 4\ell$. Then for each u_i^k, S_j compute ρ_i^k for sampler Sp_k. If any of ρ_i^k for $1 \leq k \leq 4\ell$ is greater than τ, the site sends a_i along with all its priority values, and discards it; otherwise, site S_j stores a_i, and each sampler Sp_k for $1 \leq k \leq 4\ell$ maintains a pointer to a_i and stores ρ_i^k, say, Sp_k *refers* to a_i. On receiving a row a_i and its priorities, the coordinator also stores a_i and each sampler in the coordinator refers to a_i.

Then we consider two cases where the global threshold τ is updated. Firstly, when each of all 4ℓ samplers maintains at least 2 samples with priority values greater than τ, the coordinator increases τ, and move samples in Sp_k with priorities no greater than τ to Sp_k', a candidate set for Sp_k. The new τ is set so that the number of samplers that hold at least 2 samples is exactly 2ℓ. Then τ is broadcast to m sits. Secondly, when there are only ℓ samplers containing at least 2 samples, the coordinator keep halving the threshold τ, broadcasting the new tau, and retrieving valid rows among m sites and the candidate sets of samplers, until there are at least 2ℓ samplers containing at least 2 samples. Note that on receiving each halved τ, site S_j sends the active rows with at least one priority value greater than τ, along with their indices of success.

Create estimates. We set $\ell = \frac{1}{\varepsilon^2}$ as in row sampling without replacement. The sharing-threshold protocol ensures at least ℓ samplers with at least 2 samples. Assume the top-2 samples of a qualified sampler Sp_k are (a_1, ρ_1^k) and (a_2, ρ_2^k) with $\rho_1^k \geq \rho_2^k$, we rescale a_1^k to make its squared norm equal to $\max\{\|a_1^k\|^2, \rho_2^k\}$, and combine the rescaled a_1^k of all ℓ qualified samplers to make the estimation.

Analysis. The correctness of sharing-threshold protocol is obvious. Our protocol ensures there are at least ℓ samplers each maintaining its top-2 samples in terms of priorities. Thus we omit the proof.

In the following analysis, for simplicity, we call a sampler *qualified* if it maintains at least 1 sample (instead of maintaining at least 2 samples). Obviously, any asymptotic bound derived as follows can simulate the real bound for sharing-threshold protocol with a multiplicative factor of const.

We simply analysis the space usage in each site. We use the similar strategy to reduce space usage in both the sites and the coordinator as for lazy-broadcast protocol. Apparently, in site S_j, a row a_i is useless to sampler Sp_k if a_i is right 1-dominated with respect to Sp_k, i.e., there are at least one later row a_q that has greater priorities ρ_q^k than ρ_i^k. Furthermore, row a_i can be safely discarded if it is right 1-dominated with respect to all samplers. According to Lemma 2 (take ℓ as 1), each sampler refers to expected $O(\log(NR))$ rows, and thus the union of all rows referred by all 4ℓ samplers has expected size $O(\ell\log(NR))$. For simplicity, here we ignore the dependency between samplers referring to each specific row, which doesn't affect the space complexity. We also remove all rows that expire. The coordinator applies the same space-reducing strategy as well.

Lemma 5. *For a sequence of rows $\{a_1, \cdots a_N\}$ each assigned with a priority value ρ_i as in priority sampling. Given any τ, define $f(a_i, \tau)$ as the number of index k where $\rho_i^k > \tau$, let $B_1 = \{a_i | f(a_i, \tau) \geq 1\}$, $B_0 = \{a_i | f(a_i, \tau/2) \geq 1\}$. Then,*

$$\Pr[|B_0| \geq 4|B_1|] \leq e^{-\Omega(|B_0|)}.$$

The above lemma roughly says, when there are less than $\ell\log\ell$ rows having at least 1 priority value greater than τ, then, with high probability, the number of rows with at least 1 priority value greater than $\tau/2$ cannot be too large. This is intuitively correct, since conditioned on a row has at least 1 priority value greater than $\tau/2$, it has at least 1 priority value greater than τ with good probability.

Proof: For $a_i \in B_0$, w.l.o.g, assume $\rho_i^{k'} > \tau/2$, we have the following inequality:

$$\Pr[\rho_i^{k'} > \tau | \rho_i^{k'} > \tau/2] \geq 1/2,$$

which follows directly from 1. Then we have

$$\begin{aligned} \Pr[f(a_i,\tau) \geq 1 | f(a_i, \tau/2) \geq 1] &\geq \Pr[\rho_i^{k'} > \tau | f(a_i, \tau/2) \geq 1] \\ &= \Pr[\rho_i^{k'} > \tau | \rho_i^{k'} > \tau/2] \\ &\geq 1/2. \end{aligned}$$

Thus, for each $a_i \in B_0$, we have $\Pr[a_i \in B_i] \geq 1/2$. By a standard use of Chernoff bound, we have

$$\Pr[|B_1| \leq |B_0|/4] \leq e^{-|B_0|},$$

which proves the lemma. ■

Lemma 6. *During any time interval $(t, t+W]$, the communication cost of the sharing-threshold protocol is $O(d\ell\log\ell\log(NR) + m\log(NR))$, and the number of updates of τ is $O(\log(NR))$.*

Consider the communication cost for sending rows, we make a claim here: at any fixed time t', assume τ is the global threshold, then with high probability, there are at most $16\ell\log\ell$ active rows that have at least 1 priority value greater than τ. Given this claim, using the same argument as in Lemma 3, we can conclude the communication cost of sharing-threshold protocol is $O(\ell\log\ell\log(NR))$. Next we will prove the above claim.

When τ does not change at t', the number of qualified samplers is between ℓ and 4ℓ. Conditioned on an row a_i being sent to coordinator when the threshold is τ, then a_i is equally likely to referred by any of ℓ samplers. We simply assume a_i is referred by exactly 1 sampler, and thus all the references of ell samplers are independent (with respect to different rows). According to the "balls in bins" theorem, the number of different rows sent to the coordinator is at most $8\ell\log\ell$ with high probability. When τ increases at t', then by definition, there are only 2ℓ qualified samplers, and similarly, the number of different rows referred by at least one of these 2ℓ samplers is at most $4\ell\log\ell$ with high probability. Then we consider the case where τ is just halved. At that time, there might be many active rows with at least one priority value greater than τ. However, with high probability, this cannot happen. To see that, we know τ is halved to $\tau/2$ only when there are less than 2ℓ samplers maintaining at least 1 active rows with priority values, i.e., with high probability, there are at most $4\ell\log\ell$ rows having at least 1 priority higher than τ. According to Lemma5 (with $|B_1| \leq 4\ell\log\ell$), there are at most $16\ell\log\ell$ active rows that have at least 1 priority value greater than τ with high probability (i.e., $|B_0| \leq 16\ell\log\ell$).

Now we bound the number of broadcasts, i.e., the number of times τ is updated. Consider two consecutive increases of τ at t_1 and t_2 respectively. At $t_1 + 1$, the number of qualified samplers is 2ℓ, and at t_2, the number of qualified samplers is 4ℓ. We consider two cases. If τ does not decrease between t_1 and t_2, then no pointer from the candidate set of each sampler will be added back to the sample set of sampler. Therefore, by "balls in bins" theorem with Chernoff bound, at least $\ell\log\ell$ rows are sent from sites during the time period. Otherwise, if τ decreases at some time $t' \in (t_1, t_2)$, then there are at least $\frac{1}{2}\ell\log\ell$ rows have expired during (t_1, t'). During a time window, the total number of rows sent is $O(\ell\log\ell\log(NR))$ and the total number of rows get expired is $O(\ell\log\ell\log(NR))$, and thus the number of times that τ increases is at most $O(\log(NR))$.

Then consider the moment when the number of qualified samplers becomes ℓ, i.e., τ needs to be halved. Assume β is the minimum weight among all rows, and thus the maximum weight is $R\beta$. We claim that, with high probability, $\tau < 8NR\beta$. Assume we have $\tau \geq 8NR\beta$, then for any $1 \leq i \leq N$ and $1 \leq k \leq 4\ell$, we have

$$\Pr[\rho_i^k \geq 8NR\beta] \leq 1/8N.$$

Hence for each sampler Sp_k, the probability of Sp_k being qualified is less than $1 - (1 - \frac{1}{8N})^N \approx 1 - e^{-1/8}$. For all 4ℓ samplers, the number of qualified samplers is less than $8(1 - e^{-1/8})\ell < \ell$ with high probability accordingly to Chernoff bound. Therefore $\tau < 8NR\beta$ with high probability. Note that we can ignore the dependency of samplers in above analysis by simply assuming each a_i has the highest squared norm R.

On the other hand, the minimum possible $\tau = \beta$. Thus each time the number of qualified samplers becomes ℓ, it incurs at most $\log(8NR)$ times of halving and broadcasting τ. After this, the number of qualified samplers is at least 2ℓ, which means before the next time when the number of qualified samplers becomes ℓ, there are at least $\frac{1}{2}\ell\log\ell$ rows expiring. Therefore, the times that the number of qualified sampler becomes ℓ is $O(\log(NR))$ (since we have shown at most $O(\ell\log\ell\log(NR))$ rows expire during a time window). Finally, we conclude τ is updated for at most $O(\log^2(NR))$ times during a time window, which can be reduced to $O(\log(NR))$ with a more careful analysis.

This result doesn't improve the error bounds or communication bounds with respect to row sampling without replacement. Also in terms of running time (without parallelism at each site), sampling without replacement achieves better result.

B. ES Sampling

Sampling without Replacement (ESWOR). In ES sampling, each row a_i is assigned with a random priority $u_i^{\frac{1}{\|a_i\|^2}}$, where u_i is a random variable drawn uniformly from $(0,1)$. To sample ℓ items without replacement, we also pick the ℓ items with highest priority values. Therefore, the protocol for ES sampling follows the same framework as for the priority sampling. With some minor modification, the analysis for priority sampling directly works for ES sampling.

To get a covariance sketch, each sampled row also needs to be rescaled properly: each row a_i in the sample set is rescaled by a factor of $\frac{\sqrt{\ell}\|a_i\|}{\|A_w\|_F}$ [17]. One technical issue of this method is we need to know $\|A_w\|_F^2$. Fortunately a $(1+\varepsilon)$ approximation to $\|A_w\|_F^2$ is good enough, which is equivalent to the problem of tracking sum. Tracking sum can be solved either by priority sampling above, or a deterministic method to be presented in Section III-A. Compared to the cost of row sampling, the additional cost to track $\|A_w\|_F^2$ by both methods is negligible.

Sampling with Replacement (ESWR). ES sampling with replacement follows the same framework as priority sampling with replacement.

III. Deterministic methods

In this section, we will introduce two deterministic methods for tracking covariance matrix sketch. To illustrate the idea of our first method, we will first give a deterministic protocol for tracking $\|A_w\|_F^2$, or equivalently tracking sum over a weighted sliding window stream, which generalizes and simplifies a protocol in [24] for tracking count.

A. Deterministic SUM tracking

Definition 2. In the SUM tracking problem, each item a_i in the stream has a weight w_i, the goal is to track an estimator to the sum of weights of all active items at the coordinator.

Note that the SUM tracking problem is a special case of matrix tracking (d=1). In addition, when all weights are 1, this becomes tracking COUNT, i.e., the total number of items, which was studied in [22] [24]. In particular, [24] proposed a general *forward-backward* frameworks, which solves the COUNT problem with optimal communication cost. Let $C^{(i)}(t)$ be number of active items on site S_i at time t, then $C(t) = \sum_i^m C^{(i)}(t)$ is the exact answer to the COUNT problem. To track $C(t)$ within relative ε error, it is sufficient to track each $C^{(i)}(t)$ within ε error. The forward-backward framework [24] is for tracking each individual $C^{(i)}(t)$. Together with some rounding argument as in [19], the weighted version can also be solved by forward-backward tracking.

A more direct and efficient approach for tracking SUM. A more intuitive way is to track the difference between $C^{(i)}(t)$ and $\hat{C}^{(i)}(t)$, where $\hat{C}^{(i)}(t)$ is the current SUM estimator the coordinator holds (note the site S_i also knows $\hat{C}^{(i)}(t)$) and $C^{(i)}(t)$ is the actual SUM the coordinator wishes to track. For notational convenience, we omit the superscripts and simply write $C^{(i)}$ as C and $\hat{C}^{(i)}$ as $\hat{C}$ (but C and $\hat{C}$ actually refer to actual SUM and SUM estimator with respect to a single site).

Algorithm 3 Improved Protocol for Tracking SUM

```
1: procedure TRACK_SUM(time $t$)                    ▷ at site $j$
2:     for $(w_i, t_i) = receive\_data(t)$ do
3:         Insert $(w_i, t_i)$ into $gEH$;
4:     $C = gEH.query()$.
5:     if $|C - \hat{C}| > \varepsilon C$ then
6:         send $D = C - \hat{C}$;
7:         $\hat{C} = C$.
8: procedure RECEIVE_UPDATE($D$)                    ▷ at coordinator
9:     $\hat{C} = \hat{C} + D$.
```

More precisely, S_i maintains $D = C - \hat{C}$, and whenever $|D| > \varepsilon C$, the site sends D to the coordinator and updates the estimator $\hat{C} = C$. The correctness is straightforward, since at any time t, we have $|C - \hat{C}| \leq \varepsilon C$ by definition.

The naive way to maintain exact C (or D) needs $O(N)$ space, where N is the total number of items in the sliding window. However, maintaining C exactly is not necessary, and we only need to track an ε-approximation C' to C, which can be done in $O(\frac{1}{\varepsilon}\log(WR))$ space by using *generalized exponential histogram* (gEH) [19]. Note that using ε-approximation C' instead of the exact sum C doesn't affect the correctness: adjusting ε by a constant factor in the beginning, we still promise an ε relative error for tracking SUM.

Analysis. Although, intuitively, our new protocol should have low communication cost, it is tricky to prove it rigorously.

Theorem 1. Algorithm 3 solves the SUM tracking problem over distributed sliding windows, which incurs $O(\frac{m}{\varepsilon}\log(NR))$ words of communication per window. The space usage is $O(\frac{1}{\varepsilon}\log(NR))$ words per site.

Proof: The correctness has been proved above, so we analyze the number of messages sent by a single site S_i. We divide the messages sent by S_i into two types: positive updates $(D > 0)$ and negative updates $(D < 0)$. For simplicity we assume C is exact (i.e., without using gEH). The case where C is only an ε relative approximation can be analyzed similarly with more care.

Let us consider any fixed time interval $(t - 2W, t]$. We call time window $(t - 2W, t - W]$ the *left window*, and call $(t - W, t]$ the *right window*. Let $C_l(t_{now})$ be the sum of active items in the left window at time t_{now}, and $C_r(t_{now})$ be the sum of active items in the right window.

We first bound the number of positive updates. Consider any two consecutive positive updates which happen at t_1 and t_2 respectively. By definition, $C(t_2) - C(t_1) > \varepsilon C(t_2)$, where $C(t)$ denote the value of the variable C at time t in the algorithm. This means the sum of weights of items arrive in the time interval $[t_1, t_2]$ is at least $\varepsilon C(t_2)$, i.e., C_r increases by $\varepsilon C(t_2)$. Clearly $\varepsilon C(t_2) \geq \varepsilon C_r(t_2)$, and thus C_r increases by a factor of $1 + \varepsilon$ during any two consecutive positive updates. Since C_r is bounded by NR, the number of positive updates is at most $\log_{(1+\varepsilon)}(NR) = O(\frac{1}{\varepsilon}\log(NR))$. The number of negative updates can be bounded similarly.

The space used by each site is dominated by the gEH, which is $O(\frac{1}{\varepsilon}\log(WR))$. Since the query and update time of gEH is $O(1)$ [19], the update time of the algorithm is $O(1)$. ■

Using similar ideas as in [24], **Algorithm** 3 can be extended to tracking *heavy hitters* and *quantiles*, which will not be discussed in this paper. Instead, we will show how to use this idea to track matrix.

B. First deterministic protocols for tracking covariance sketch

Let $A_w^{(j)}$ be the matrix consisting of active rows at site S_j, so the concatenation of these matrices $A_w = [A_w^{(1)};\cdots;A_w^{(m)}]$ is the target matrix. If we can track each $A_w^{(j)}$ separately within covariance error ε, i.e., the coordinator has a matrix $B^{(j)}$ for each j such that $\|A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\| \le \varepsilon\|A_w^{(j)}\|_F^2$, then matrix $B = [B^{(1)};\cdots;B^{(m)}]$ is a ε covariance sketch with respect to A_w. To see that, we have

$$\begin{aligned}\|A_w^TA_w - B^TB\| &= \|\sum_{i=1}^{m}\left(A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\right)\| \\ &\le \sum_{j=1}^{m}\|A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\| \\ &\le \sum_{j=1}^{m}\varepsilon\|A_w^{(j)}\|_F^2 = \varepsilon\|A_w\|_F^2,\end{aligned}$$

where we used triangle inequality for spectral norm in the first inequality above. Therefore, we only consider the problem of tracking the sliding window matrix on each site separately, and for notational convenience, we omit the superscripts and simply write $A_w^{(j)}$ as A_w (but it actually means the sliding window matrix at site j).

High level idea of DA1. First, assume each site is allowed to store all rows received in the current time window. We use the same "template" as in SUM tracking, except now C and $\hat{C}$ are $d\times d$ matrices. Let $C{=}A_w^TA_w$ and $\hat{C}{=}B^TB$, where B is the covariance sketch of A_w maintained by the coordinator. Site S_i maintains C and $D{=}C-\hat{C}$ exactly, and whenever $\|D\| \ge \varepsilon\|A_w\|_F^2$, S_i reports the variation. The key difference here is that S_j does not send the whole D, but sends the "significant directions" of D only, so that the remaining difference is tolerable. Otherwise, the communication cost is unbounded.

Let $D = \sum_{i=1}^{d}\lambda_i v_i^Tv_i$ be the standard eigen-decomposition of D, i.e., each v_i is an eigenvector (as a row vector) of D corresponding to the eigenvalue λ_i. The site S_i sends (λ_j, v_j) for all j such that $|\lambda_j| > \varepsilon\|A_w\|_F^2$, and updates $\hat{C} = \hat{C} + \lambda\cdot v_i^Tv_i$. For each (λ_j, v_j) being received, the coordinator updates its $\hat{C}$ accordingly. Now we have $\|A_w^TA_w - \hat{C}\| = \|C-\hat{C}\| \le \varepsilon\|A_w\|_F^2$ on the coordinator. To get a ε-covariance sketch to A_w, we compute the *matrix square root* B of $\hat{C}$, i.e., $\hat{C} = B^TB$ [33].[2]

Maintaining C (or D) exactly needs to store all active rows. As in the SUM tracking protocol, here we will use the *matrix*

[2] B exists, since $\hat{C}$ on the coordinator is *positive semidefinite*, which can be computed with SVD.

Algorithm 4 DA1 for Tracking Matrix Approximation

```
1:  procedure TRACK_MATRIX(time $t$)                          ▷ at site $j$
2:      for row $(a_i, t_i)$ = $receive\_row(t)$ do
3:          Insert $(a_i, t_i)$ into $mEH$;
4:      $(C, \|\hat{A}_w\|_F^2) = mEH.query()$;
5:      $D = C - \hat{C}$.
6:      if $\|D\| > \varepsilon\|\hat{A}_w\|_F^2$ then
7:          Compute $D = \sum_{i=1}^{d}\lambda_i v_i^Tv_i$.    ▷ Compute eigen-decomposition of $D$
8:          for $|\lambda_i| \ge \varepsilon\|\hat{A}_w\|_F^2$ do
9:              $\hat{C} = \hat{C} + \lambda\cdot v_i^Tv_i$;
10:             Send $(\lambda_i, v_i)$.
11: procedure RECEIVE_UPDATE($\lambda, v_i$)                  ▷ at coordinator
12:     $\hat{C} = \hat{C} + \lambda\cdot v_i^Tv_i$.
13: procedure QUERY()                                          ▷ at coordinator
14:     $(U, \Sigma, V) = \mathsf{SVD}(\hat{C})$;
15:     return $\Sigma^{1/2}V^T$
```

exponential histogram (mEH) of [17] to save space. The mEH takes $O(\frac{d}{\varepsilon^2}\log(NR))$ words of space, and always maintains an ε-covariance sketch C' to $A_w^TA_w$. Besides, mEH maintains an ε relative estimate $\|\hat{A}_w\|_F^2$ to $\|A_w\|_F^2$. By adjusting ε by a constant factor in the beginning, we still promise an ε covariance error with our approach. For simplicity, we use C instead of C' in all illustration.

The pseudocode of DA1 is shown in Algorithm 4. By a similar (but more technical) analysis as in the SUM tracking protocol, we can show that the communication cost per window is $O(\frac{md}{\varepsilon}\log(NR))$.

Lemma 7. *In DA1 the communication cost per window is $O(\frac{md}{\varepsilon}\log(NR))$.*

Proof: For simplicity, we only prove the case when $C = A_w^TA_w$ (i.e., without using matrix exponential histogram). The general case can be proved similarly. We divide the messages sent by S_i into two types: positive updates $(\lambda > 0)$ and negative updates $(\lambda < 0)$. Next we bound the number of positive updates, while the negative updates can be analyzed similarly.

We consider the time window $(t, t+W]$. We divide the matrix A_w (rows in $(t_{now} - W, t_{now}]$) into two submatrices A_e and A_a, where A_e is the expiring matrix, i.e., the rows arrived during $(t_{now} - W, t]$, and A_a is the active matrix, i.e., the rows arrived during $(t, t_{now}]$. We divide the time window into two tracking periods: the first period ends at the first time when $\|A_e\|_F \le \|A_a\|_F$.

We divide the first period into rounds; in each round $\|A_e\|_F^2$ is halved. We claim the number of rows sent is at most $O(1/\varepsilon)$ in a round. To see that, Assume the eigenvalues sent during this round is $\lambda^{(1)},\cdots,\lambda^q$. At the end of each round, the norm of A_w is at most doubled, and thus $\sum\lambda^{(j)} = O(\|A_w\|_F^2)$. In this round, we have $\lambda^{(j)} \ge \varepsilon\|A_w\|_F^2$ for each j, which implies that $q = O(1/\varepsilon)$. The total number of rounds is $O(\log(NR))$, hence the communication cost in the first period is $O(d\log(NR)/\varepsilon)$ for each site. Using the same argument, the cost of the second period is also $O(d\log(NR)/\varepsilon)$. ■

The space usage is dominated by mEH, which is

$O(\frac{d}{\varepsilon^2}\log(NR))$.

Limitations of DA1. However, there are a few limitations to the above deterministic algorithm. First, we need to compute eigen-decomposition of a $d \times d$ matrix (i.e., D) frequently, each of which takes $O(d^3)$ time. Secondly, even with matrix exponential histogram, it requires additional d^2 space and $O(d^2)$ time per update to maintain D. Hence, the algorithm is not suitable for large d. Motivated by this, we propose a second deterministic algorithm DA2 with better scalability in terms of d. Empirical comparisons of two deterministic algorithms will be the experiments section.

High level idea of DA2. The most time-consuming part of DA1 is computing the eigen-decomposition of D. In DA2, we will use the *frequent direction* (FD) algorithm of [12] to dynamically maintain an approximate of D. The challenge is that D takes negative updates, i.e., expiration of old rows, while FD can only handle positive updates. The idea of DA2 is motivated by the forward-backward framework [24]. The whole tracking period is divided into windows of length W: $(0, W], (W, 2W], \cdots, (iW, (i+1)W], \cdots$. Assume the current time is $kW < t_{now} < (k+1)W$, the target matrix is a concatenation of two sub-matrices, namely $A_e(t_{now})$ and $A_a(t_{now})$, corresponding the *expiring window* $(t_{now}-W, kW)$ and *active window* (kW, t_{now}) respectively. When there is no confusion, we simply write them as A_e and A_a. As the time window $(t_{now} - W, t_{now}]$ slides, new rows are added to A_a while old rows expire from A_e. We then track A_a and A_e separately with ε covariance error. Tracking A_a is essentially the same as the infinite window case, which can be solved by a protocol from [1] combined with FD (for efficiency). However, instead of tracking A_e directly, we track $M(t_{now})$, the matrix in $((k-1)W, t_{now}-W)$. Since we have $A_e(kW) = [M(t_{now}); A_e(t_{now})]$ for any $t_{now} \in (kW, (k+1)W]$, and an ε-covariance sketch of $A_e(kW)$ is already known by coordinator at time kW, thus given $M(t_{now})$, the coordinator can compute an ε-covariance sketch to $A_e(t_{now})$.

In the following, we first assume each site stores all active rows. Similar as in DA1, we can use matrix exponential histogram to save space. Although the original infinite window tracking protocol (P2) in [1] needs two-way communication, here we only use this for tracking a single site, so it is essentially a one-way protocol. We call this IWMT protocol. We regard IWMT as a black box, which receives as input a sequence of rows and sends/outputs another row sequence; the covariance error between two matrices formed by any prefixes of the two row sequences is bounded by a given threshold.

We simply use a IWMT protocol to track A_a (forward tracking), namely $IWMT_a$. By stacking all rows received so far from $IWMT_a$, the coordinator has a sketch $\hat{A}_a(t)$ which is an ε-covariance sketch of $A_a(t)$. We next discuss how to track A_e (backward tracking). At time t_{now}=kW for some integer k, site S_j starts a IWMT protocol, namely $IWMT_c$, on $A_e(kW)$ in the reverse time direction locally, and uses a queue Q to record every message (m_i, t_i) from $IWMT_c$, where m_i is a row and t_i is the timestamp[3]. Recall that we actually want to track $M(t)$. To do so, site S_j starts another IWMT protocol at $t = kW$, namely $IWMT_e$, which does a forward tracking on Q according to expiring time, i.e., at any time, we feed each expired row in Q to $IWMT_e$.

More formally, let $Q = \{(m_i, t_i) \mid i = 1, 2, \cdots, \}$ be the messages recorded by applying $IWMT_c$ on $A_e(kW)$ in the reverse order of time. Since site S_j has $A_e(kW)$ at time kW, Q can be computed by simulating IWMT on the row stream formed by the rows in $A_e(kW)$ in the reverse order. Then, $IWMT_e$ reads the expired rows from Q as t_{now} increases, and sends messages to the coordinator if necessary.

Note in IWMT protocol [1], a threshold is used to limit the maximum norm of unsent content. Basically, the larger the threshold is, the less number of messages are sent. We set the thresholds for three IWMT protocols in our algorithm as follow: (1) the input of $IWMT_c$ is a row sequence of descending timestamp (in a reverse order), and on receiving (a_i, t_i), $IWMT_c$ uses $\varepsilon\|A_e(t_i + W)\|_F^2$ as threshold, i.e., the total squared norms of rows received by $IWMT_c$ thus far. This follows exactly the original IWMT protocol. (2) both $IWMT_a$ and $IWMT_e$ use $\varepsilon\|A_w\|_F^2$ as the threshold, for $t \in (kW, (k+1)W)$. This is actually an *optimal* threshold, thus the overall communication cost of DA2 can potentially be much lower compared to a basic forward-backward protocol.

It is not so straightforward to see the reason why we need $IWMT_c$ and $IWMT_e$ to track the matrix expiring window, instead of using a single $IWMT_e$ protocol to read the row stream of $A_e(kW)$ directly. We will explain the reason as follows. As mentioned, we use $\varepsilon\|A_w\|_F^2$ as the threshold for $IWMT_e$ while tracking the matrix in expiring window. If we had $IWMT_e$ read the row stream of $A_e(kW)$ directly, the sketch maintained by $IWMT_e$, we denote it by $\hat{M_e}(t)$ could either follow FD protocol or not. If it follows FD protocol, then the overall covariance error cannot be bounded by ε. Notice the covariance error between $\hat{M_e}(t)$ and $M_e(t)$ is bounded by $\varepsilon\hat{M_e}(t)$ always, which however, could be greater than $\varepsilon\|A_e\|_F^2$. Consider the case where there is no row in the active window, i.e., $\|A_w\|_F^2 = \|A_e\|_F^2$. With FD protocol, each time being truncated, $\hat{M_e}(t)$ discards some *minor* directions with respect to $\varepsilon\|\hat{M_e}(t)\|_F^2$, which will surely be greater than $\|A_e\|_F^2$ at a time $(k-1)W \leq t' \leq kW$, hence the coordinator will not approximate A_e with ε covariance error. If the sketch $\hat{M_e}(t)$ does not follow FD protocol, notice the size of rows stream of $A_e(kW)$ is $O(N)$, and thus maintaining $\hat{M_e}(t)$ can be extremely costly with respect to both time-consuming and space complexity. Motivated by above analysis, we employee $IWMT_c$ and $IWMT_e$ to tracking A_e, i.e., we use $IWMT_c$ to *compress* the row streams of $A_e(kW)$ to a size of $O(\varepsilon \log NR)$ rows, thus the sketch $\hat{M_e}(t)$ of $IWMT_e$ can still process updates fast with sublinear space, even it does not follow FD.

For completeness, we give a lemma for the correctness of IWMT protocol.

Lemma 8. *Matrix $A(t)$ is the concatenation of rows observed thus far based on the time-based row sequence S_1. Matrix $B(t)$ is the concatenation of rows observed thus far from another time-based row sequence S_2, where S_2 is the output sequence of IWMT protocol with input S_1 and threshold $\|A(t)\|_F^2$. At any timestamp t', $\|A(^t{}')TA('t) - B^T(t')B(t')\|_2 \leq \varepsilon\|A(t')\|_F^2$.*

[3] $IWMT_c$ process A_e in the reverse order, but the timestamp of each message is recorded using the actual time, and thus the messages in Q is ordered in the reverse order of t

The correctness of Lemma 8 has been proved in [1]. Then we will prove the correctness of DA2.

We denote by S^o the row sequence of $A_e(kW)$ stored by mEH structure, $A_e^o(t)$ the matrix concatenated by rows in S^o with timestamps greater than $t - W$. We denote by S^c the row sequence output by $IWMT_c$ with input S^o in a reverse order and threshold $\varepsilon\|A_e^o(t)\|_F^2$, $A_e^c(t)$ the matrix concatenated by rows in S^c with timestamps greater than $t-W$, $M_e^c(t)$ the matrix satisfying $A_e^c(t)^T A_e^c(t) + M_e^c(t)^T M_e^c(t) = A_e(kW)^T A_e(kW)$ (the exact $A_e(kW)^T A_e(kW)$ is maintained by coordinator with $d \times d$ space). We denote by S^e the row sequence output by $IWMT_e$ with input S^o in ascending order of timestamp and threshold $\varepsilon\|\hat{A}_w(t)\|_F^2$, $M_e^e(t)$ the matrix concatenated by rows in S^e with timestamps smaller than $t - W$. Note that the coordinator computes $A_e(kW)^T A_e(kW) - M_e^e(t)^T M_e^e(t)$ as the covariance sketch to $A_e(t)^T A_e(t)$, and we will prove the covariance error is bounded by $4\varepsilon\|\hat{A}_w(t)\|_F^2$ for $kW < t \leq (k+1)W$.

According to Lemma 8, we have

$$\|A_e^c(t)^T A_e^c(t) - A_e^o(t)^T A_e^o(t)\|_2 \leq \varepsilon\|A_e^o(t)\|_F^2,$$

and

$$\|M_e^e(t)^T M_e^e(t) - M_e^c(t)^T M_e^c(t)\|_2 \leq \varepsilon\|\hat{A}_w(t)\|_F^2.$$

By combining the above inequalities, we have

$$\begin{aligned}&\|A_e^c(t)^T A_e^c(t) - A_e^o(t)^T A_e^o(t) - M_e^e(t)^T M_e^e(t) + M_e^c(t)^T M_e^c(t)\|_2 \\ &= \|A_e(kW)^T A_e(kW) - M_e^e(t)^T M_e^e(t) - A_e^o(t)^T A_e^o(t)\| \\ &\leq \varepsilon(\|A_e^o(t)\|_F^2 + \|A_w(t)\|_F^2).\end{aligned}$$

Then we consider the relationship between $A_e^o(t)$ and $A_e(t)$. By the property of mEH, we have

$$\|A_e^o(t)^T A_e^o(t) - A_e(t)^T A_e(t)\|_2 \leq \varepsilon\|A_e(t)\|_F^2.$$

and finally

$$\begin{aligned}&\|A_e(kW)^T A_e(kW) - M_e^e(t)^T M_e^e(t) - A_e(t)^T A_e(t)\| \\ &\leq \varepsilon(\|A_e^o(t)\|_F^2 + \|A_e(t)\|_F^2 + \|A_w(t)\|_F^2) \\ &\leq 4\varepsilon\|A_w(t)\|_F^2.\end{aligned}$$

The worst-case communication cost during the time window $(kW, (k+1)W)$ is no more than twice the cost of IWMT for tracking a single site, which is $O(\frac{d}{\varepsilon}\log(NR))$ [1], and thus the total communication cost is $O(\frac{md}{\varepsilon}\log(NR))$ per window. The amortized update time of the IWMT protocol is $O(\frac{d}{\varepsilon})$ if we use frequent direction of [12] to speed up the computation. If we use matrix exponential histogram to save space, the space usage will be $O(\frac{d}{\varepsilon^2}\log(NR))$ [17].

IV. Experiments

A. Experiments Setting

Data sets. To evaluate our algorithms, we use two publicly available data sets "PAMAP" and "WIKI", as well as a synthetic data set in the experiment. The data sets are summarised in Table III.

Algorithm 5 DA2 for Tracking Matrix Approximation

```
1:  procedure TRACK_MATRIX(time $t_{now}$)                ▷ at site $j$
2:      for row $a_i$ = $receive\_row(t_{now})$ do
3:          $m_i = IWMT_a(a_i, t_{now})$; Send $(m_i, 1)$;
4:          Insert $(a_i, t_{now})$ into $mEH_a$;
5:      if $t_{now} == kW$ for integer $k$ then        ▷ End of window
6:          $Q = IWMT_c(mEH_a)$.
7:      for $(m_i, t_i) \in Q$ and $t_i < t_{now} - W$ do
8:          $m'_i = IWMT_e(m_i, t_i)$; Send $(m'_i, -1)$;
9:          Remove $(m_i, t_i)$ from $Q$.
10: procedure RECEIVE_UPDATE($m_i$,$flag$)           ▷ at coordinator
11:     $\hat{C} = \hat{C} + flag \times m_i^T m_i$.
12: procedure QUERY()                                ▷ at coordinator
13:     $(U, \Sigma, V) = \mathsf{SVD}(\hat{C})$;
14:     return $\Sigma^{1/2} V^T$
```

PAMAP[4] is a Physical Activity Monitoring dataset, which is the data of 18 different physical activities (such as walking, cycling, playing soccer, etc.), performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor. The dataset contains 54 columns including a timestamp, an activity label (the ground truth) and 52 attributes of raw sensory data. In our experiments, we used a subset with $N = 814729$ rows and $d = 43$ columns (removing columns containing missing values). We generate synthetic timestamps for PAMAP following the *Poisson Arrival Process* [34] with $\lambda = 1$. Moreover, we set the window size such that on average there are approximately $200,000$ rows in a window.

WIKI[5]is the text corpus built on the article dump of the September 2015 version of English Wikipedia. The words occurring at least 1000 times in the entire corpus are used as features (columns), and the articles with at least 500 features are selected as rows. The entry at row i and column j is the tf-idf weight of article i and word j. The timestamp of each row is the date when the article is published, i.e., we view one day as a time unit. The matrix consists of 78608 rows and 7047 columns. The timestamp spans over 3949 days. We set the window size to 502 such that on average there are 10000 rows in a window.

SYNTHETIC is a random noisy matrix that have been commonly used for evaluating matrix sketching algorithms [17] [1] [12]. The matrix is concatenated by 3 submatrices of the same size. Each submatrix is generated by formula $A{=}SDU{+}N/\zeta$, where S is a $n \times d$ coefficients matrix with each entry drawn from standard normal distribution, D is a diagonal matrix with $D_{i,i}{=}1{-}(i{-}1)/d$, U is a random matrix satisfying $UU^T{=}I_d$, N contributes additive Gaussian noise with each entry drawn from standard norm distribution, and ζ is assigned 10 thus the real signal is recoverable. The matrix has 300 columns and $500,000$ rows. We generate synthetic timestamps for SYNTHETIC, following the *Poisson Arrival Process* [34] with $\lambda = 1$. We set the window size such that on average there are approximately $100,000$ rows in a window.

Algorithms. We compared all of the protocols proposed in this paper, including sampling based and deterministic tracking algorithms.

[4]http://www.pamap.org/demo.html

[5]https://en.wikipedia.org/wiki/Wikipedia:Database_download

Data Sets	total rows n	d	average rows per window	ratio R
PAMAP	814,729	43	$\approx$ 200,000	60.78
SYNTHETIC	500,000	300	$\approx$ 100,000	3.72
WIKI	78,608	7047	$\approx$ 10,000	2998.83

Table III: Summary of Data sets.

(1) Priority sampling. We evaluated the performance of priority sampling without replacement, denoted as PWOR. Recall that the coordinator maintains a "candidate" sample set with at least ℓ samples. Intuitively, the accuracy will be improved by using all available samples, which are actually "free". Based on this observation, we also implemented a variation of PWOR, denoted as PWOR-ALL, which makes use of all samples available to the coordinator.

(2) ES sampling. We evaluated ESWOR, the ES sampling algorithms for without-replacements scheme, and ESWOR-ALL, the ES sampling algorithms that makes use of all "candidate samples". ES sampling differs from priority sampling by using different priority functions and estimators, and thus it is interesting to compare their performance on real world data sets.

(3) Deterministic. We evaluated the DA1 and DA2 algorithms. We remark that DA1 is very slow for large dimension d, and could not finish on WIKI, so we only present the results of DA1 on PAMAP and SYTHETIC.

Note that we excluded the with-replacement sampling algorithms from our experiments. Firstly, sampling with replacement is extremely time-consuming and thus is infeasible for processing large matrix data. Secondly, as observed in [1] and [17], SWOR is at least as accurate as SWR on most datasets.

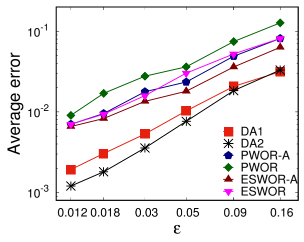
Metrics. In the experimental study, we used default value $\varepsilon = 0.05$ and $m = 20$. Note that for all of our protocols, the theoretical analyses only show asymptotic worst-case communication costs, which rarely happen in real data sets. To evaluate above algorithms, we tested the trade-off between actual communication cost and observed covariance error. We also measured the communication cost and accuracy as ε and m varied, as well as the update rate and space usage of each site. The metrics are defined as follows:

1) Communication cost *msg* is defined to be the average number of words sent per window. We assume each real number takes 1 word.
2) Approximation error *err* is defined to be $\|A_w^T A_w - B^T B\|_2 / \|A_w\|_F^2$, where A_w is the matrix of current sliding window and B is the matrix approximation. We randomly picked 50 query points and measured both average and maximum error (denoted by avg_err and max_err).

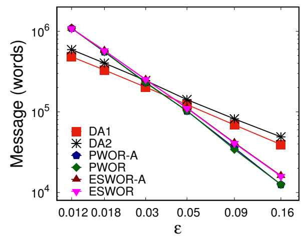
Setup. For random sampling, we ran the algorithms for 3 times in each experiment, and reported the average communication cost and error over 3 executions. All algorithms were implemented in Python 3.5.1 using 64-bit addressing. The experiments were executed on Intel Xeon E3-1231V3, clocked at 3.4 GHz.

B. Performance Study

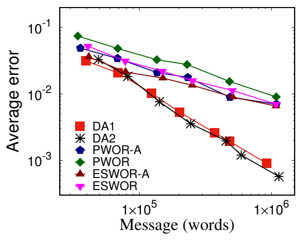
Error vs. Communication cost. We make the following observations:



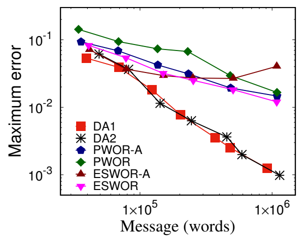
(a) avg_err vs. ε.



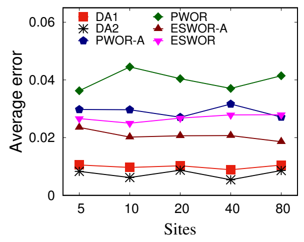
(b) msg vs. ε.



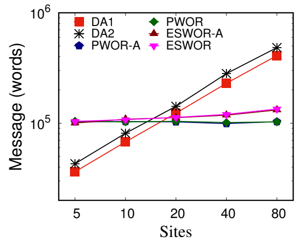
(c) avg_err vs. msg.



(d) max_err vs. msg.

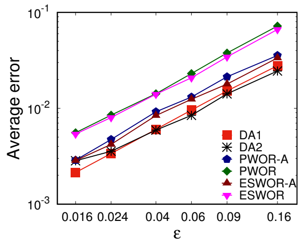


(e) avg_err vs. sites.

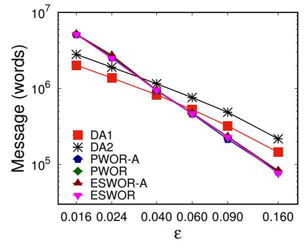


(f) msg vs. sites.

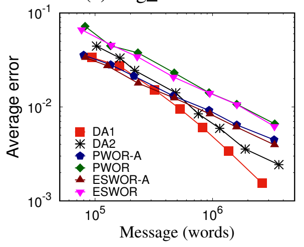
Figure 1: Results on PAMAP dataset.



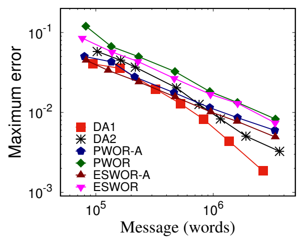
(a) avg_err vs. ε.



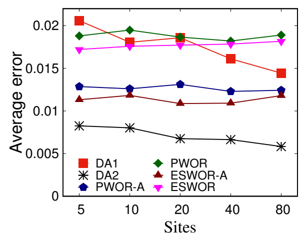
(b) msg vs. ε.



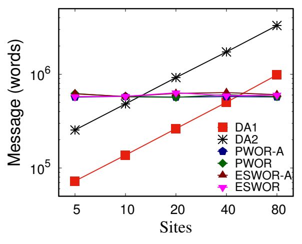
(c) avg_err vs. msg.



(d) max_err vs. msg.

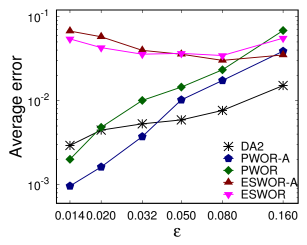


(e) avg_err vs. sites.

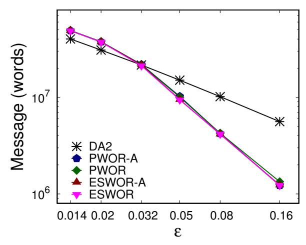


(f) msg vs. sites.

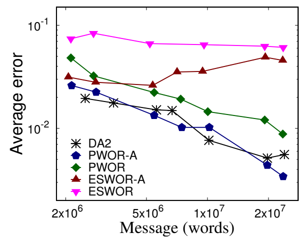
Figure 2: Results on SYNTHETIC dataset.



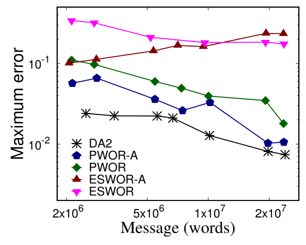
(a) avg_err vs. ε on 20 sites.



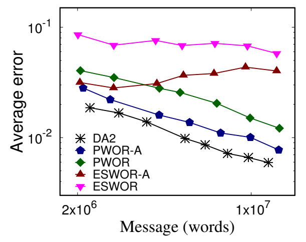
(b) msg vs. ε on 20 sites.



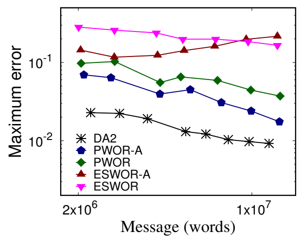
(c) avg_err vs. msg on 20 sites.



(d) max_err vs. msg on 20 sites.



(e) avg_err vs. msg on 10 sites.



(f) max_err vs. msg on 10 sites.

Figure 3: Results on WIKI dataset.

(1) Figure 1(a), 2(a), and 3(a) show the tradeoffs between the error parameter ε and the average actual error of the protocols. In most cases, the observed error for all protocols is smaller than the parameter ε. We also observe that deterministic protocols provides better error guarantee than sampling methods do, under the same error parameter ε. Inside the sampling family, using all available samples typically achieves better accuracy.

(2) Figure 1(b), 2(b), and 3(b) show the tradeoffs between the error parameter ε and the communication cost of protocols. It is worth noticing that as ε decreases, the communication cost of deterministic protocols grows much slower than random sampling does, which confirms the dependency of their asymptotic bound on ε ($1/\varepsilon$ vs. $1/\varepsilon^2$). Communication cost of ES sampling is slightly higher than that of priority sampling, which is a result of the extra cost of tracking $\|A_w\|_F^2$.

(3) The tradeoffs between observed error and communication cost are shown in Figure 1(c),1(d), Figure 2(c),2(d), and Figure 3(c),3(d) respectively. In the default setting where $m = 20$, DA1 and DA2 achieve better "communication vs error" tradeoff than that of the sampling methods. The advantage of the deterministic algorithms is more significant for maximum error. This is as expected, since the error guarantee of sampling methods is randomized. For the two deterministic protocols, we have the following observations: (i) DA1 is notably better on SYNTHETIC dataset, where rows of the matrix are generated from the same distribution; (ii) The two protocols have similar performance on PAMAP; (iii) DA1 is too slow to finish the experiments on WIKI. For sampling methods, making use of all available samples (PWOR-ALL and ESWOR-ALL) usually achieves better tradeoff than top-ℓ samples (PWOR and ESWOR). Furthermore, PWOR-ALL significantly outperforms ESWOR-ALL on WIKI dataset, while ESWOR-ALL has slight advantage on PAMAP dataset. They both behave well on SYNTHETIC dataset.

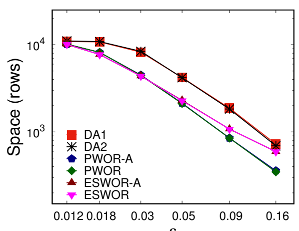
(4) Another interesting observation is that, on skewed datasets (PAMAP and WIKI) (i.e. R is large), increasing the communication does not necessarily reduce the error of ESWOR-ALL (Figure 1(d), 3(c) and 3(d)). One possible explanation is that ES sampling rescales each sample a_i by a factor of $\frac{\|A_w\|_F}{\sqrt{\ell'}\|a_i\|}$, after which all samples have the same squared norm. As a result, a sampled row with small norm will be greatly stretched once sampled, and is over-emphasized in final estimation. When the size of sample set increases, such "bad" events occur more frequently and lead to relative high covariance error. This conjecture is also supported by Figure 3(c) and 3(d), which show that the maximum error of ESWOR-ALL is higher than that of ESWOR when communication cost exceeds 10^7 words. We believe that this is because ESWOR only takes top-ℓ samples, and thus decreases the risk of picking rows with small norms.

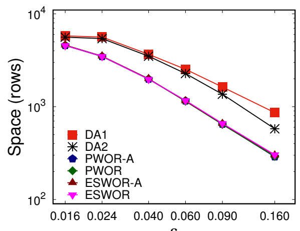
On the other hand, priority sampling rescales a_i to make its squared norm equal to $\max\{\|a_i\|^2, \tau_\ell\}$, which sets a *ceiling* τ_ℓ for rows with small norms while ensuring the contribution of rows with large norms. Hence we recommend priority sampling rather than ES sampling on skewed datasets.

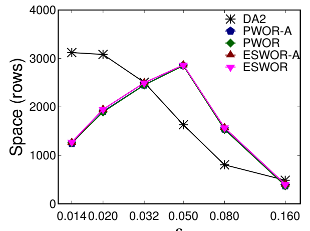
Varying number of sites. We also conducted experiments to show how the error and communication cost changes as the number of sites varies. We set the number of sites for PAMAP and SYNTHETIC to vary from 5 to 80. The total number of rows is relatively small for WIKI, so we only tested the performance for 10 and 20 sites to make sure that each site receives enough rows.

We make the following observations: (1) The covariance error of all protocols is stable as m varies (Figure 1(e) and 2(e)); (2) For sampling methods, the communication cost remains the same as m increases (Figure 1(f) and 2(f)). This is consistent with our analysis, since the communication cost is dominated by the term $O(\frac{d}{\varepsilon^2}\log(NR))$, which does not depend on m. (3) The communication cost of deterministic protocols clearly has a linear dependence on the number of sites, which matches our theoretical analysis (both DA1 and DA2 need $O(\frac{md}{\varepsilon}\log(NR))$ words of communication). In general, the sampling protocols are scalable with the site number while the deterministic algorithms are not.

Space usage and update rate. Finally, we measured the space usage and update rate of all protocols. Figures 4(a)4(b)4(c) shows the tradeoffs between maximum space usages and the error parameter ε of all protocols. When ε is small, all protocols store almost all rows observed in the window. In most cases, the space usages of all protocols are close to each other, which confirms our theoretical analysis: the space needed by each site is roughly $\widetilde{O}(\frac{d}{\varepsilon^2})$ words for both deterministic and sampling protocols. DA1 requires extra $d \times d$ space to maintain matrix sketches, which is negligible when d is small (PAMAP and SYNTHETIC). On the WIKI dataset, however, the space usage of DA2 does not decrease significantly as ε increases. The reason is that the large ratio R (2998.83 to be exact) of WIKI dataset greatly limits the compression effect of the


(a) space vs. ε on PAMAP.


(b) space vs. ε on SYNTHETIC.


(c) space vs. ε on WIKI.

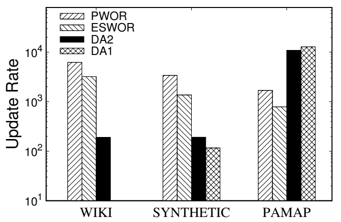

(d) Update Rate.

Figure 4: Space usage and update rate vs. ε on all datasets.

MEH structure. We also note that the space usage of sampling methods is small when ε is set to be small on WIKI dataset. This is because that each site sends many rows to coordinator for the large sample set, and thus the number of rows left in each site is greatly decreased.

Figure 4(d) shows the update rates of protocols (updates processed per second) under the default setting where $\varepsilon = 0.05$ and $m = 20$. We observe that the deterministic protocols process updates faster than sampling methods do for low-dimensional matrix (PAMAP), while their update rate decreases dramatically as d increases. This is because DA1 and DA2 need to compute matrix factorizations periodically, which leads to running time quadratic or even cubic in d. For WIKI dataset (where $d \approx 7000$), DA1 is too slow to finish the experiments, while the update rate for DA2 is 10 to 100 times slower than sampling methods. Notice that the update rate of sampling methods is not affected by d. Interestingly, the sampling algorithms process update faster on WIKI than on PAMAP. This is reasonable, since d only affects the time to compute the norm, which is not the dominating part of the running time.

C. Remarks

We conclude our experimental evaluations with a few remarks. In general, the deterministic protocols achieves better "communication vs. error" tradeoff than sampling methods. Moreover, the deterministic protocols achieve worst-case error guarantee, which is desirable when one wants bounded maximum error over the whole stream. So if high accuracy is the main concern, the deterministic protocols are recommended. It is also worth mentioning that deterministic protocols only use one-way communication and no global synchronization is needed. DA1 and DA2 have similar performance in terms of accuracy and communication, while DA2 outperforms DA1 in terms of processing time. Therefore, we recommend DA2 for large d for its time-efficiency, and recommend DA1 when d is small as it is much easier to implement.

On the other hand, the sampling methods trade accuracy for some other desirable properties. For instance, sampling methods produce a sketch consists of rows of the original matrix, which improves interpretability and preserves row structure. This is essential for some applications such as *column/row subset selection* [35]. Moreover, the processing time for the sampling methods process is much faster than that of the deterministic algorithms when the dimension d is large. Inside the sampling schemes, the overall performance of PWOR-ALL is the best, and thus we recommend PWOR-ALL for matrix approximation.

V. Conclusion

In this paper, we initialize the study of continuously tracking a matrix approximation over distributed sliding windows, and propose protocols with low communication cost and space usage for the problem. We first consider row sampling methods, and propose novel algorithms for tracking weighted random samples over distributed sliding windows, which generalize and simplify previous works for unweighted sampling. We also provide two deterministic protocols which only use one-way communication and have a better dependence on ε in terms of communication cost. In addition to theoretical analysis, extensive experiments are conducted on large-scale real and synthetic data sets, and a detailed comparison of their performances is provided.

References

[1] M. Ghashami, J. M. Phillips, and F. Li, "Continuous matrix approximation on distributed data," *Proceedings of the VLDB Endowment*, 2014.

[2] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Continuous sampling from distributed streams," *Journal of the ACM (JACM)*, vol. 59, no. 2, p. 10, 2012.

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

[4] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 4, p. 23, 2007.

[5] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," *ACM Transactions on Algorithms (TALG)*, vol. 7, no. 2, p. 21, 2011.

[6] Z. Huang, K. Yi, and Q. Zhang, "Randomized algorithms for tracking distributed count, frequencies, and ranks," in *Proceedings of PODS*, 2012.

[7] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster, "Prediction-based geometric monitoring over distributed data streams," in *Proceedings of SIGMOD*, 2012.

[8] M. B. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proceedings of PODS*, 2004.

[9] A. Manjhi, S. Nath, and P. B. Gibbons, "Tributaries and deltas: Efficient and robust aggregation in sensor network streams," in *Proceedings of SIGMOD*. ACM, 2005, pp. 287–298.

[10] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[11] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *ACM SIGMETRICS Performance evaluation review*, vol. 32, no. 1, 2004, pp. 61–72.

[12] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of SIGKDD*, 2013.

[13] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *Proceedings of VLDB*, 2005.

[14] M. Ghashami and J. M. Phillips, "Relative errors for deterministic low-rank matrix approximations," in *Proceedings of SODA*, 2014.

[15] H. Huang and S. P. Kasiviswanathan, "Streaming anomaly detection using randomized matrix sketching," *Proceedings of the VLDB Endowment*, 2015.

[16] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff, "Frequent directions: Simple and deterministic matrix sketching," *SIAM Journal on Computing*, vol. 45, no. 5, pp. 1762–1792, 2016.

[17] Z. Wei, X. Liu, F. Li, S. Shang, X. Du, and J.-R. Wen, "Matrix sketching over sliding windows," in *Proceedings of SIGMOD*, 2016.

[18] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall, "A wakeup call for internet monitoring systems: The case for distributed triggers," in *Proceedings of HotNets-III*, 2004.

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

[20] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proceedings of SODA*, 2002.

[21] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of PODS*, 2002.

[22] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting, "Continuous monitoring of distributed data streams over a time-based sliding window," *Algorithmica*, vol. 62, no. 3-4, pp. 1088–1111, 2012.

[23] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, "Sketch-based querying of distributed sliding-window data streams," *Proceedings of the VLDB Endowment*, 2012.

[24] G. Cormode and K. Yi, "Tracking distributed aggregates over time-based sliding windows," in *International Conference on Scientific and Statistical Database Management*, 2012.

[25] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *Proceedings of SIGKDD*. ACM, 2015, pp. 935–944.

[26] C. Boutsidis, D. Garber, Z. Karnin, and E. Liberty, "Online principal components analysis," in *Proceedings of SODA*, 2015.

[27] Z. Karnin and E. Liberty, "Online pca with spectral bounds," in *Proceedings of COLT*, 2015, pp. 505–509.

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

[29] N. Duffield, C. Lund, and M. Thorup, "Priority sampling for estimation of arbitrary subset sums," *Journal of the ACM (JACM)*, vol. 54, no. 6, p. 32, 2007.

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

[31] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *arXiv preprint arXiv:1411.4357*, 2014.

[32] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[33] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[34] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Poisson_point_process

[35] J. A. Tropp, "Column subset selection, matrix factorization, and eigenvalue optimization," in *Proceedings of SODA*, 2009.