Continuous Matrix Approximation on Distributed Data*

Mina Ghashami
School of Computing
University of Utah
ghashami@cs.uah.edu

Jeff M. Phillips
School of Computing
University of Utah
jeffp@cs.uah.edu

Feifei Li
School of Computing
University of Utah
lifeifei@cs.uah.edu

Tracking and approximating data matrices in streaming fashion is a fundamental challenge. The problem requires more care and attention when data comes from multiple distributed sites, each receiving a stream of data. This paper considers the problem of "tracking approximations to a matrix" in the distributed streaming model. In this model, there are m distributed sites each observing a distinct stream of data (where each element is a row of a distributed matrix) and has a communication channel with a coordinator, and the goal is to track an ε-approximation to the norm of the matrix along any direction. To that end, we present novel algorithms to address the matrix approximation problem. Our algorithms maintain a smaller matrix B, as an approximation to a distributed streaming matrix A, such that for any unit vector x: $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon\|A\|_F^2$. Our algorithms work in streaming fashion and incur small communication, which is critical for distributed computation. Our best method is deterministic and uses only $O((m/\varepsilon)\log(\beta N))$ communication, where N is the size of stream (at the time of the query) and β is an upper-bound on the squared norm of any row of the matrix. In addition to proving all algorithmic properties theoretically, extensive experiments with real large datasets demonstrate the efficiency of these protocols.

1. INTRODUCTION

Large data matrices are found in numerous domains, such as scientific computing, multimedia applications, networking systems, server and user logs, and many others [27–29, 37]. Since such data is huge in size and often generated continuously, it is important to process them in streaming fashion and maintain an approximating summary. Due to its importance, the matrix approximation problem has received careful investigations in the literature; the latest significant effort is represented by Liberty [29] on a centralized stream.

In recent years, *distributed streaming model* [14] has become popular, in which there are multiple distributed sites, each observing a disjoint stream of data and together attempt to monitor a function at a single coordinator site C. Due to its wide applications in practice [14], a flurry of work has been done under this setting. This model is more general than the *streaming model* [3] that maintains a function at a single site with small space. It is also different from *communication model* [43] in which data is (already) stored at multiple sites and the goal is to do a *one-time* computation of a target function. The key resources to optimize in distributed streaming model is not just the space needed at the coordinator or each site, but the communication between the sites and coordinator.

Despite prior works on distributed streaming model, and distributed matrix computations (e.g., the MadLINQ library [39]), little is known on continuously tracking an matrix approximation in the distributed streaming model. This paper considers the important problem of *"tracking approximations to matrices"* in the distributed streaming model [14].

Motivation. Our problem is motivated by many applications in distributed databases, wireless sensor networks, cloud computing, etc [35] where data sources are distributed over a network and collecting all data together at a central location is not a viable option. In many such environments queries must be answered continuously, based on the total data that has arrived so far.

For example, in large scale image analysis, each row in the matrix corresponds to one image and contains either pixel values or other derived feature values (e.g, 128-dimensional SIFT features). A search engine company has image data continuously arriving at many data centers, or even within a single data center at many nodes in a massive cluster. This forms a distributed matrix and it is critical to obtain excellent, real-time approximation of the distributed streaming image matrix with little communication overhead.

Yet another example is for large-scale distributed web crawling or server access log monitoring/mining, where data in the bag-of-words model is a matrix whose columns correspond to words or tags/labels (for textual analysis, e.g. LSI, and/or for learning and classification purpose) and rows correspond to documents or log records (which arrive continuously at distributed nodes).

Since data is continuously changing in these applications, query results can also change with time. So the challenge is to minimize the communication between sites and the coordinator while maintaining accuracy of results *at all time*.

In our case, each site may generate a record in a given time instance, which is a row of a matrix. The goal is to *approximate the matrix that is the union of all the rows from*

*Thanks to support from NSF grants CCF-1115677, IIS-1251019, IIS-1200792, and IIS-1251019.

Proceedings of the VLDB Endowment, Vol. 7, No. 10

all sites until current time instance t_{now} continuously at the coordinator C. This problem can be easily found in distributed network monitoring applications [38], distributed data mining, cloud computing [39], stream mining [24], and log analysis from multiple data centers [9].

Distributed streaming matrix approximation. Formally, assume $A = (a_1, \ldots, a_n, \ldots)$ is an unbounded stream of items. At the current time t_{now}, let n denote the number of items the system has seen so far; that is at the current time the dataset is $A = (a_1, \ldots, a_n)$. And although we do not place a bound on the number of items, we let N denote the total size of the stream *at the time when a query q is performed*. This allows us to discuss results in terms of n at a given point, and in terms of N for the entire run of the algorithm until the time of a query q.

At each time step we assume the item a_n appears at exactly one of m sites $S_1, \ldots, S_m$. The goal is to approximately maintain or monitor some function $f(A)$ of A at the coordinator node C. Each site can only communicate with the coordinator (which in practice may actually be one of the sites). This model is only off by a factor 2 in communication with a model where all pairs of sites can communicate, by routing through the coordinator.

Our goal is to maintain a value $\hat{f}(A)$ which is off by some ε-factor from the true value $f(A)$. The function and manner of approximation is specified for a variety of problems [4, 11, 12, 26, 31, 44] included total count, heavy hitters, quantiles, and introduced in this paper, matrix approximation:

Definition 1 (Tracking distributed streaming matrix) : Suppose each a_n is a record with d attributes, a row from a matrix in an application. Thus, at time t_{now}, $A = (a_1, \ldots, a_n)$ forms a $n \times d$ *distributed streaming matrix*. At any time instance, C needs to approximately maintain the norm of matrix A along any arbitrary direction. The goal is to continuously track a small approximation of matrix A. Formally, for any time instance t_{now} (i.e., for any n), C needs to maintain a smaller matrix $B \in \mathbb{R}^{\ell \times d}$ as an approximation to the distributed streaming matrix $A \in \mathbb{R}^{n \times d}$ such that $\ell \ll n$ and for any unit vector x: $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2$.

As required in the distributed streaming model [14], each site S_i must process its incoming elements in streaming fashion. The objective is to minimize the total communication between C and all sites $S_1, \ldots, S_m$. $\square$

Additional notations. In the above definition, the Frobenius norm of a matrix A is $\|A\|_F = \sqrt{\sum_{i=1}^{n} \|a_i\|^2}$ where $\|a_i\|$ is standard Euclidean norm of row a_i; the Frobenius norm is a widely used matrix norm. We also let A_k be the *best rank k approximation* of matrix A, specifically $A_k = \arg\min_{X:\mathsf{rank}(X) \leq k} \|A - X\|_F$.

Our contributions. This work makes important contributions in solving the open problem of tracking distributed streaming matrix. Instead of exploring heuristic methods that offer no guarantees on approximation quality, we focus on principled approaches that are built on sound theoretical foundations. Moreover, all methods are simple and efficient to implement as well as effective in practice. Specifically:

- We establish and further the important connection between tracking a matrix approximation and maintaining ε-approximate *weighted* heavy-hitters in the distributed streaming model in Section 4. Initial insights along this direction were established in very recent work [21, 29]; but the extent of this connection is still limited and not fully understood. This is demonstrated, for instance, by us showing how three approaches for weighted heavy hitters can be adapted to matrix sketching, but a fourth cannot.
- We introduce four new methods for tracking the ε-approximate *weighted* heavy-hitters in a distributed stream in Section 4, and analyze their behaviors with rigorous theoretical analysis.
- We design three novel algorithms for tracking a good approximation of a distributed streaming matrix in Section 5; these leverage the new insights connecting this problem to solutions in Section 4 for distributed weighted heavy hitters tracking.
- We present thorough experimental evaluations of the proposed methods in Section 6 on a number of large real data sets. Experimental results verify the effectiveness of our methods in practice.

In addition, we provide a thorough review of related works and relevant background material in Section 2. The paper is concluded in Section 7.

2. RELATED WORK

There are two main classes of prior work that are relevant to our study: approximating a streaming matrix in a centralized stream, and tracking heavy hitters in either a centralized stream or (the union of) distributed steams.

Matrix approximation in a centralized stream. Every incoming item in a centralized stream represents a new row of data in a streaming matrix. The goal is to continuously maintain a low rank matrix approximation. It is a special instance of our problem for $m = 1$, i.e., there is only a single site. Several results exist in the literature, including streaming PCA (principal component analysis) [34], streaming SVD (singular value decomposition) [7, 40], and matrix sketching [8, 21, 29]. The matrix sketching technique [29] only recently appeared and is the start-of-the-art for low-rank matrix approximation in a single stream. Liberty [29] adapts a well-known streaming algorithm for approximating item frequencies, the MG algorithm [33], to sketching a streaming matrix. The method, Frequent Directions (FD), receives n rows of a matrix $A \in \mathbb{R}^{n \times d}$ one after another, in a centralized streaming fashion. It maintains a sketch $B \in \mathbb{R}^{\ell \times d}$ with only $\ell \ll n$ rows, but guarantees that $A^T A \approx B^T B$. More precisely, it guarantees that $\forall x \in \mathbb{R}^d$, $\|x\| = 1$, $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq 2\|A\|_F^2/\ell$. FD uses $O(d\ell)$ space, and each item updates the sketch in amortized $O(d\ell)$ time; two such sketches can also be merged in $O(d\ell^2)$ time.

A bound on $\|Ax\|$ for any unit vector x preserves norm (or length) of a matrix in direction x. For instance, when one performs PCA on A, it returns the set of the top k orthogonal directions, measured in this length. These are the linear combinations of attributes (here we have d such attributes) which best capture the variation within the data in A. Thus by increasing ℓ, this bound allows one to approximately retain all important linear combinations of attributes.

An extension of FD to derive streaming sketch results with bounds on relative errors, i.e., to ensure that $\|A - A_k\|_F^2 \leq \|A\|_F^2 - \|B_k\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$, appeared in [21]. It also gives that $\|A - \pi_{B_k}(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$ where B_k is the top k rows of B and $\pi_{B_k}(A)$ is the projection of A onto the row-space of B_k. This latter bound is interesting because, as we will see, it indicates that when most of the variation is captured in the first k principal components, then we can almost recover the entire matrix exactly.

But none of these results can be applied in distributed streaming model without incurring high communication cost. Even though the FD sketches are mergeable, since the coordinator C needs to maintain an approximation matrix *continuously* for all time instances. One has to either send m sketches to C, one from each site, at every time instance and ask C to merge them to a single sketch, or one can send a streaming element to C whenever it is received at a site and ask C to maintain a single sketch using FD at C. Either approach will lead to $\Omega(N)$ communication.

Nevertheless, the design of FD has inspired us to explore the connection between distributed matrix approximation and approximation of distributed heavy hitters.

Heavy hitters in distributed streams. Tracking heavy hitters in distributed streaming model is a fundamental problem [4,26,31,44]. Here we assume each $a_n \in A$ is an element of a bounded universe $[u] = \{1, \ldots, u\}$. If we denote the frequency of an element $e \in [u]$ in the stream A as $f_e(A)$, the ϕ-heavy hitters of A (at time instance t_{now}) would be those items e with $f_e(A) \geq \phi n$ for some parameter $\phi \in [0, 1]$. We denote this set as $H_\phi(A)$. Since computing exact ϕ-heavy hitters incurs high cost and is often unnecessary, we allow an ε-approximation, then the returned set of heavy hitters must include $H_\phi(A)$, may or may not include items e such that $(\phi - \varepsilon)n \leq f_e(A) < \phi n$ and must not include items e with $f_e(A) < (\phi - \varepsilon)n$.

Babcock and Olston [4] designed some deterministic heuristics called as *top-k monitoring* to compute top-k frequent items. Fuller and Kantardzid modified their technique and proposed *FIDS* [20], a heuristic method, to track the heavy hitters while reducing communication cost and improving overall quality of results. Manjhi *et.al.* [31] studied ϕ-heavy hitter tracking in a hierarchical communication model.

Cormode and Garofalakis [11] proposed another method by maintaining a summary of the input stream and a prediction sketch at each site. If the summary varies from the prediction sketch by more than a user defined tolerance amount, the summary and (possibly) a new prediction sketch is sent to a coordinator. The coordinator can use the information gathered from each site to continuously report frequent items. Sketches maintained by each site in this method require $O((1/\varepsilon^2)\log(1/\delta))$ space and $O(\log(1/\delta))$ time per update, where δ is a probabilistic confidence.

Yi and Zhang [44] provided a deterministic algorithm with communication cost $O((m/\varepsilon)\log N)$ and $O(1/\varepsilon)$ space at each site to continuously track ϕ-heavy hitters and the ϕ-quantiles. In their method, every site and the coordinator have as many counters as the type of items plus one more counter for the total items. Every site keeps track of number of items it receives in each round, once this number reaches roughly ε/m times of the total counter at the coordinator, the site sends the counter to the coordinator. After the coordinator receives m such messages, it updates its counters and broadcasts them to all sites. Sites reset their counter values and continue to next round. To lower space usage at sites, they suggested using space-saving sketch [32]. The authors also gave matching lower bounds on the communication costs for both problems, showing their algorithms are optimal in the deterministic setting.

Later, Huang *et.al.* [23] proposed a randomized algorithm that uses $O(1/(\varepsilon\sqrt{m}))$ space at each site and $O((\sqrt{m}/\varepsilon)\log N)$ total communication and tracks heavy hitters in a distributed stream. For each item a in the stream a site chooses to send a message with a probability $p = \sqrt{m}/(\varepsilon\hat{n})$ where $\hat{n}$ is a 2-approximation of the total count. It then sends $f_e(A_j)$ the total count of messages at site j where $a = e$, to the coordinator. Again an approximation heavy-hitter count $\hat{f}_e(A_j)$ can be used at each site to reduce space.

The ε-heavy hitters can be maintained from a random sampling of elements of size $s = O(1/\varepsilon^2)$. This allows one to use the well studied technique of maintaining a random sample of size s from a distributed stream [15,42], which can be done with roughly $O((m + s)\log(N/s))$ communication.

Other related work. Lastly, our work falls into the general problem of tracking a function in distributed streaming model. Many existing works have studied this general problem for various specific functions, and we have reviewed the most related ones on heavy hitters. A detailed survey of results on other functions (that are much less relevant to our study) is beyond the scope of this work, and we refer interested readers to [10, 14] and references therein.

Our study is also related to various matrix computations over distributed matrices, for example, the MadLINQ library [39]. However, these results focus on one-time computation over non-streaming data, i.e., in the *communication model* [43] as reviewed in Section 1. We refer interested readers to [39] and references therein for details.

There are also many studies on low-rank approximations of matrices in centralized, non-streaming setting, e.g., [1, 16, 19, 30] and others, but these methods are not applicable for a distributed streaming setting.

3. INSIGHTS, WEIGHTS AND ROUNDS

A key contribution of this paper is strengthening the connection between maintaining approximate weighted frequency counts and approximately maintaining matrices.

To review, the Misra-Gries (MG) algorithm [33] is a deterministic, associative sketch to approximate frequency counts, in contrast to, say the popular count-min sketch [13] which is randomized and hash-based. MG maintains an associative array of size ℓ whose keys are elements of $e \in [u]$, and values are estimated frequency $\hat{f}_e$ such that $0 \leq f_e - \hat{f}_e \leq n/\ell$. Upon processing element $e \in A$, three cases can occur. If e matches a label, it increments the associated counter. If not, and there is an empty counter, it sets the label of the counter to e and sets its counter to 1. Otherwise, if no empty counters, then it decrements (shrinks) all counters by 1.

Liberty [29] made this connection between frequency estimates and matrices in a centralized stream through the singular value decomposition (svd). Our approaches avoid the svd or use it in a different way. The svd of an $n \times d$ matrix A returns (among other things) a set of d singular values $\{\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d\}$ and a corresponding set of orthogonal right singular vectors $\{v_1, \ldots, v_d\}$. It holds that $\|A\|_F^2 = \sum_{i=1}^d \sigma_i^2$ and for any x that $\|Ax\|^2 = \sum_{i=1}^d \sigma_i^2 \langle v_i, x\rangle^2$. The work of Liberty [29] shows that one can run a version of the Misra-Gries [33] sketch for frequency counts using the singular vectors as elements and squared singular values as the corresponding total weights, recomputing the svd when performing the shrinkage step.

To see why this works, let us consider the restricted case where every row of the matrix A is an indicator vector along the standard orthonormal basis. That is, each row $a_i \in \{e_1, \ldots, e_d\}$, where $e_j = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the jth standard basis vector. Such a matrix A can encode a stream S of items from a domain $[d]$. If the ith element in the stream

S is item $j \in \{1, \ldots, d\}$, then the ith row of the matrix A is set to $a_i = e_j$. At time t_{now}, the frequency f_j from S can be expressed as $f_j = \|Ae_j\|^2$, since $\|Ax\|^2 = \sum_{i=1}^{n} \langle a_i, x \rangle^2$ and the dot product is only non-zero in this matrix for rows which are along e_j. A good approximate matrix B would be one such that $g_j = \|Be_j\|^2$ is a good approximation of f_j. Given $\|A\|_F^2 = n$ (since each row of A is a standard basis vector), we derive that $|f_j - g_j| \leq \varepsilon n$ is equivalent to $|\|Ae_j\|^2 - \|Be_j\|^2| \leq \varepsilon \|A\|_F^2$.

However, general matrices deviate from this simplified example in having non-orthonormal rows. Liberty's FD algorithm [29] demonstrates how taking svd of a general matrix gets around this deviation and it achieves the same bound $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2$ for any unit vector x.

Given the above connection, in this paper, first we propose four novel methods for tracking *weighted heavy hitters* in a distributed stream of items (*note that tracking* weighted *heavy hitters in the distributed streaming model has not been studied before*). Then, we try to extend them to track matrix approximations where elements of the stream are d-dimensional vectors. Three extensions are successful, including one based directly on Liberty's algorithm, and two others establishing new connections between weighted frequency estimation and matrix approximation. We also show why the fourth approach cannot be extended, illustrating that the newly established connections are not obvious.

Upper bound on weights. As mentioned above, there are two main challenges in extending ideas from frequent items estimation to matrix approximation. While, the second requires delving into matrix decomposition properties, the first one is in dealing with weighted elements. We discuss here some high-level issues and assumptions we make towards this goal.

In the weighted heavy hitters problem (similarly in matrix approximation problem) each item a_i in the stream has a weight w_i (for matrices this weight will be implicit as $\|a_i\|^2$). Let $W = \sum_{i=1}^{n} w_i$ be the total weight of the problem. However, allowing arbitrary weights can cause problems as demonstrated in the following example.

Suppose we want to maintain a 2-approximation of the total weight (i.e. a value $\hat{W}$ such that $\hat{W} \leq W \leq 2\hat{W}$). If the weight of each item doubles (i.e. $w_i = 2^i$ for tuple $(a_i, w_i) \in A$), every weight needs to be sent to the coordinator. This follows since W more than doubles with every item, so $\hat{W}$ cannot be valid for more than one step. The same issue arises in tracking approximate heavy hitters and matrices.

To make these problems well-posed, often researchers [25] assume weights vary in a finite range, and are then able to bound communication cost. To this end we assume all $w_i \in [1, \beta]$ for some constant $\beta \geq 1$.

One option for dealing with weights is to just pretend every item with element e and weight w_i is actually a set of $\lceil w_i \rceil$ distinct items of element e and weight 1 (the last one needs to be handled carefully if w_i is not an integer). But this can increase the total communication and/or runtime of the algorithm by a factor β, and is not desirable.

Our methods take great care to only increase the communication by a $\log(\beta N)/\log N$ factor compared to similar unweighted variants. In unweighted version, each protocol proceeds in $O(\log N)$ rounds (sometimes $O(\frac{1}{\varepsilon} \log N)$ rounds); a new round starts roughly when the total count n doubles. In our settings, the rounds will be based on the total weight W, and will change roughly when the total weight W doubles. Since the final weight $W \leq \beta N$, this will causes an increase to $O(\log W) = O(\log(\beta N))$ rounds. The actual analysis requires much more subtlety and care than described here, as we will show in this paper.

4. WEIGHTED HEAVY HITTERS IN A DISTRIBUTED STREAM

The input is a *distributed* weighted data stream A, which is a sequence of tuples $(a_1, w_1), (a_2, w_2), \ldots, (a_n, w_n), \ldots$ where a_n is an element label and w_n is the weight. For any element $e \in [u]$, define $A_e = \{(a_i, w_i) \mid a_i = e\}$ and let $W_e = \sum_{(a_i, w_i) \in A_e} w_i$. For notational convenience, we sometimes refer to a tuple $(a_i, w_i) \in A$ by just its element a_i.

There are numerous important motivating scenarios for this extension. For example, instead of just monitoring counts of objects, we can measure a total size associated with an object, such as total number of bytes sent to an IP address, as opposed to just a count of packets.

We next describe how to extend four protocols for heavy hitters to the weighted setting. These are extensions of the unweighted protocols described in Section 2.

Estimating total weight. An important task is to approximate the current total weight $W = \sum_{i=1}^{n} w_i$ for all items across all sites. This is a special case of the heavy hitters problem where all items are treated as being the same element. So if we can show a result to estimate the weight of any single element using a protocol within εW, then we can get a global estimate $\hat{W}$ such that $|W - \hat{W}| \leq \varepsilon W$. All our subsequent protocols can run a separate process in parallel to return this estimate if they do not do so already.

Recall that the heavy hitter problem typically calls to return all elements $e \in [u]$ if $f_e(A)/W \geq \phi$, and never if $f_e(A)/W < \phi - \varepsilon$. For each protocol we study, the main goal is to ensure that an estimate $\hat{W}_e$ satisfies $|f_e(A) - \hat{W}_e| \leq \varepsilon W$. We show this, along with the $\hat{W}$ bound above, adjusting constants, is sufficient to estimate weighted heavy hitters. We return e as a ϕ-weighted heavy hitter if $\hat{W}_e/\hat{W} > \phi - \varepsilon/2$.

Lemma 1 *If $|f_e(A) - \hat{W}_e| \leq (\varepsilon/6)W$ and $|W - \hat{W}| \leq (\varepsilon/5)W$, we return e if and only if it is a valid weighted heavy hitter.*

PROOF. We need $|\frac{\hat{W}_e}{\hat{W}} - \frac{f_e(A)}{W}| \leq \varepsilon/2$. We show the upper bound, the lower bound argument is symmetric.

$$\frac{\hat{W}_e}{\hat{W}} \leq \frac{f_e(A)}{\hat{W}} + \frac{\varepsilon}{6}\frac{W}{\hat{W}} \leq \frac{f_e(A)}{W}\frac{1}{1-\varepsilon/5} + \frac{\varepsilon}{5}\frac{1+\varepsilon/5}{1-\varepsilon/5}$$
$$\leq \frac{f_e(A)}{W} + \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{f_e(A)}{W} + \frac{\varepsilon}{2}. \quad \square$$

Given this result, we can focus just on approximating the frequency $f_e(A)$ of all items.

4.1 Weighted Heavy Hitters, Protocol 1

We start with an intuitive approach to the distributed streaming problem: run a streaming algorithm (for frequency estimation) on each site, and occasionally send the full summary on each site to the coordinator. We next formalize this protocol (P1).

On each site we run the Misha-Gries summary [33] for frequency estimation, modified to handle weights, with $2/\varepsilon = 1/\varepsilon'$ counters. We also keep track of the total weight W_i of all data seen on that site i since the last communication

with the coordinator. When W_i reaches a threshold τ, site i sends all of its summaries (of size only $O(m/\varepsilon)$) to the coordinator. We set $\tau = (\varepsilon/2m)\hat{W}$, where $\hat{W}$ is an estimate of the total weight across all sites, provided by the coordinator. At this point the site resets its content to empty. This is summarized in Algorithm 4.1.

The coordinator can merge results from each site into a single summary without increasing its error bound, due to the mergeability of such summaries [2]. It broadcasts the updated total weight estimate $\hat{W}$ when it increases sufficiently since the last broadcast. See details in Algorithm 4.2.

Algorithm 4.1 P1: Tracking heavy-hitters (at site S_i)

for (a_n, w_n) in round j **do**
  Update $G_i \leftarrow \mathsf{MG}_{\varepsilon'}(G_i, (a_n, w_n))$.
  Update total weight on site W_i += w_n.
  if $(W_i \geq \tau = (\varepsilon/2m)\hat{W})$ **then**
    Send (G_i, W_i) to coordinator; make G_i, W_i empty.

Algorithm 4.2 P1: Tracking heavy-hitters (at C)

On input (G_i, W_i):
Update sketch $S \leftarrow \mathsf{Merge}_{\varepsilon'}(S, G_i)$ and W_C += W_i.
if $(W_C/\hat{W} > 1 + \varepsilon/2)$ **then**
  Update $\hat{W} \leftarrow W_C$, and broadcast $\hat{W}$ to all sites.

Lemma 2 *(P1) Algorithms 4.1 and 4.2 maintain that for any item $e \in [u]$ that $|f_e(S) - f_e(A)| \leq \varepsilon W_A$. The total communication cost is $O((m/\varepsilon^2)\log(\beta N))$ elements.*

Proof. For any item $e \in [u]$, the coordinator's summary S has error coming from two sources. First is the error as a result of merging all summaries sent by each site. By running these with an error parameter $\varepsilon' = \varepsilon/2$, we can guarantee [2] that this leads to at most $\varepsilon' W_C \leq \varepsilon W_A/2$, where W_C is the weight represented by all summaries sent to the coordinator, hence less than the total weight W_A.

The second source is all elements on the sites not yet sent to the coordinator. Since we guarantee that each site has total weight at most $\tau = (\varepsilon/2m)\hat{W} \leq (\varepsilon/2m)W$, then that is also an upper bound on the weight of any element on each site. Summing over all sites, we have that the total weight of any element not communicated to the coordinator is at most $m \cdot (\varepsilon/2m)W = (\varepsilon/2)W$.

Combining these two sources of error implies the total error on each element's count is *always* at most εW, as desired.

The total communication bound can be seen as follows. Each message takes $O(1/\varepsilon)$ space. The coordinator sends out a message to all m sites every (at most) m updates it sees from the coordinators; call this period an epoch. Thus each epoch uses $O(m/\varepsilon)$ communication. In each epoch, the size of W_C (and hence $\hat{W}$) increases by an additive $m \cdot (\varepsilon/2m)\hat{W} \geq (\varepsilon/4)W_A$, which is at least a relative factor $(1 + \varepsilon/4)$. Thus starting from a weight of 1, there are k epochs until $1 \cdot (1+\varepsilon/4)^k \geq \beta N$, and thus $k = O(\frac{1}{\varepsilon}\log(\beta N))$. So after all k epochs the total communication is at most $O((m/\varepsilon^2)\log(\beta N))$. □

4.2 Weighted Heavy-Hitters Protocol 2

Next we observe that we can significantly improve the communication cost of protocol P1 (above) using an observation, based on an unweighted frequency estimation protocol by Yi and Zhang [44]. Algorithms 4.3 and 4.4 summarize this protocol.

Each site takes an approach similar to Algorithm 4.1, except that when the weight threshold is reached, it does not send the entire summary it has, but only the weight at the site. It still needs to report heavy elements, so it also sends e whenever any element e's weight has increased by more than $(\varepsilon/m)\hat{W}$ since the last time information was sent for e. Note here it only sends that element, not all elements.

After the coordinator has received m messages, then the total weight constraint must have been violated. Since $W \leq \beta N$, at most $O(\log_{(1+\varepsilon)}(\beta N)) = O((1/\varepsilon)\log(\beta N))$ rounds are possible, and each round requires $O(m)$ total weight messages. It is a little trickier (but not too hard) to see it requires only a total of $O((m/\varepsilon)\log(\beta N))$ element messages, as follows from the next lemma; it is in general not true that there are $O(m)$ such messages in one round.

Algorithm 4.3 P2: Tracking heavy-hitters (at site S_i)

for each item (a_n, w_n) **do**
  W_i += w_n and Δ_{a_n} += w_n.
  if $(W_i \geq (\varepsilon/m)\hat{W})$ **then**
    Send (total, W_i) to C and reset $W_i = 0$.
  if $(\Delta_{a_n} \geq (\varepsilon/m)\hat{W})$ **then**
    Send (a_n, Δ_{a_n}) to C and reset $\Delta_{a_n} = 0$.

Algorithm 4.4 P2: Tracking heavy-hitters (at C)

On message (total, W_i):
Set $\hat{W}$ += W_i and $\#\mathsf{msg}$ += 1.
if $(\#\mathsf{msg} \geq m)$ **then**
  Set $\#\mathsf{msg} = 0$ and broadcast $\hat{W}$ to all sites.
On message (a_n, Δ_n): set $\hat{W}_{a_n}$ += Δ_{a_n}.

Lemma 3 *After r rounds, at most $O(m \cdot r)$ element update messages have been sent.*

Proof. We prove this inductively. Each round gets a budget of m messages, but only uses t_i messages in round i. We maintain a value $T_r = r \cdot m - \sum_{i=1}^{r} t_i$. We show inductively that $T_r \geq 0$ at all times.

The base case is clear, since there are at most m messages in round 1, so $t_1 \leq m$, thus $T_1 = m - t_1 \geq 0$. Then since it takes less than 1 message in round i to account for the weight of a message in a round $i' < i$. Thus, if $\sum_{i=1}^{r-1} t_i = n_r$, so $k_r = (r-1)m - n_r$, then if round i had more than $m + k_r$ messages, the coordinator would have weight larger than having m messages from each round, and it would have at some earlier point ended round r. Thus this cannot happen, and the inductive case is proved. □

The error bounds follow directly from the unweighted case from [44], and is similar to that for (P1). We can thus state the following theorem.

Theorem 1 *Protocol 2 (P2) sends $O(\frac{m}{\varepsilon}\log(\beta N))$ total messages, and approximates all frequencies within εW.*

One can use the space-saving algorithm [32] to reduce the space on each site to $O(m/\varepsilon)$, and the space on the coordinator to $O(1/\varepsilon)$.

4.3 Weighted Heavy-Hitters Protocol 3

The next protocol, labeled (P3), simply samples elements to send to the coordinator, proportional to their weight. Specifically we combine ideas from priority sampling [18] for without replacement weighted sampling, and distributed

sampling on unweighted elements [15]. In total we maintain a random sample S of size at least $s = O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ on the coordinator, where the elements are chosen *proportional to their weights*, unless the weights are large enough (say greater than W/s), in which case they are always chosen. By deterministically sending all large enough weighted elements, not only do we reduce the variance of the approach, but it also means the protocol naturally sends the full dataset if the desired sample size s is large enough, such as at the beginning of the stream. Algorithm 4.5 and Algorithm 4.6 summarize the protocol.

We denote total weight of sample by W_S. On receiving a pair (a_n, w_n), a site generates a random number $r_n \in \mathsf{Unif}(0,1)$ and assigns a priority $\rho_n = w_n/r_n$ to a_n. Then the site sends triple (a_n, w_n, ρ_n) to the coordinator if $\rho_n \geq \tau$, where τ is a global threshold provided by the coordinator.

Initially τ is 1, so sites simply send any items they receive to the coordinator. At the beginning of further rounds, the coordinator doubles τ and broadcasts it to all sites. Therefore at round j, $\tau = \tau_j = 2^j$. In any round j, the coordinator maintains two priority queues Q_j and Q_{j+1}. On receiving a new tuple (a_n, w_n, ρ_n) sent by a site, the coordinator places it into Q_{j+1} if $\rho_n \geq 2\tau$, otherwise it places a_n into Q_j.

Once $|Q_{j+1}| = s$, the round ends. At this time, the coordinator doubles τ as $\tau = \tau_{j+1} = 2\tau_j$ and broadcasts it to all sites. Then it discards Q_j and examines each item (a_n, w_n, ρ_n) in Q_{j+1}, if $\rho_n \geq 2\tau$, it goes into Q_{j+2}, otherwise it remains in Q_{j+1}.

Algorithm 4.5 P3: Tracking heavy-hitters (at site S_i)

for (a_n, w_n) in round j **do**
  choose $r_n \in \mathsf{Unif}(0,1)$ and set $\rho_n = w_n/r_n$.
  if $\rho_n \geq \tau$ **then** send (a_n, w_n, ρ_n) to C.

Algorithm 4.6 P3: Tracking heavy-hitters (at C)

On input of (a_n, w_n, ρ_n) from any site in round j:
if $\rho > 2\tau_j$ **then** put a_n in Q_{j+1},
  else put a_n in Q_j.
if $|Q_{j+1}| \geq s$ **then**
  Set $\tau_{j+1} = 2\tau_j$; broadcast τ_{j+1} to all sites.
  for $(a_n, w_n, \rho_n) \in Q_{j+1}$ **do**
    if $\rho_n > 2\tau_{j+1}$, put a_n in Q_{j+2}.

At any time, a sample of size exactly s can be derived by subsampling from $Q_j \cup Q_{j+1}$. But it is preferable to use a larger sample $S = Q_j \cup Q_{j+1}$ to estimate properties of A, so we always use this full sample.

Communication analysis. The number of messages sent to the coordinator in each round is $O(s)$ with high probability. To see that, consider an arbitrary round j. Any item a_n being sent to coordinator at this round, has $\rho_n \geq \tau$. This item will be added to Q_{j+1} with probability

$$\mathbf{Pr}(\rho_n \geq 2\tau \mid \rho_n \geq \tau) = \frac{\mathbf{Pr}(\rho_n \geq 2\tau)}{\mathbf{Pr}(\rho_n \geq \tau)} = \frac{\mathbf{Pr}(r_n \leq \frac{w_n}{2\tau})}{\mathbf{Pr}(r_n \leq \frac{w_n}{\tau})} = \frac{\min(1, \frac{w_n}{2\tau})}{\min(1, \frac{w_n}{\tau})} \geq \frac{1}{2}.$$

Thus sending $4s$ items to coordinator, the expected number of items in Q_{j+1} would be greater than or equal to $2s$. Using a Chernoff-Hoeffding bound $\mathbf{Pr}(2s - |Q_{j+1}| > s) \leq \exp(-2s^2/4s) = \exp(-s/2)$. So if in each round $4s$ items are sent to coordinator, with high probability (at least $1 - \exp(-s/2)$), there would be s elements in Q_{j+1}. Hence each round has $O(s)$ items sent with high probability.

The next lemma, whose proof is in the appendix section of the full version of our paper [22], bounds the number of rounds. Intuitively, each round requires the total weight of the stream to double, starting at weight s, and this can happen $O(\log(\beta N/s))$ times.

Lemma 4 *The number of rounds is at most $O(\log(\beta N/s))$ with probability at least $1 - e^{-\Omega(s)}$.*

Since with probability at least $1 - e^{-\Omega(s)}$, in each round the coordinator receives $O(s)$ messages from all sites and broadcasts the threshold to all m sites, we can then combine with Lemma 4 to bound the total messages.

Lemma 5 *This protocol sends $O((m+s)\log\frac{\beta N}{s})$ messages with probability at least $1 - e^{-\Omega(s)}$. We set $s = \Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$.*

Note that each site only requires $O(1)$ space to store the threshold, and the coordinator only requires $O(s)$ space.

Creating estimates. To estimate $f_e(A)$ at the coordinator, we use a set $S' = Q_j \cup Q_{j+1}$ which is of size $|S'| = s' > s$. Let $\hat{\rho}$ be the priority of the smallest priority element in S'. Let S be all elements in S' except for this single smallest priority element. For each of the $s'-1$ elements in S assign them a weight $\bar{w}_i = \max(w_i, \hat{\rho})$, and we set $W_S = \sum_{a_i \in S} \bar{w}_i$. Then via known priority sampling results [18, 41], it follows that $\mathbf{E}[W_S] = W_A$ and that $(1-\varepsilon)W_A \leq W_S \leq (1+\varepsilon)W_A$ with large probability (say with probability $1-\varepsilon^2$, based on variance bound $\mathbf{Var}[W_S] \leq W_A^2/(s'-2)$ [41] and a Chebyshev bound). Define $S_e = \{a_n \in S \mid a_n = e\}$ and $f_e(S) = \sum_{a_n \in S_e} \bar{w}_n$.

The following lemma, whose proof is in the appendix, of the full version [22], shows that the sample maintained at the coordinator gives a good estimate on item frequencies. At a high-level, we use a special Chernoff-Hoeffding bound for negatively correlated random variables [36] (since the samples are without replacement), and then only need to consider the points selected that have small weights, and thus have values in $\{0, \hat{\rho}\}$.

Lemma 6 *With $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$, the coordinator can use the estimate from the sample S such that, with large probability, for each item $e \in [u]$, $|f_e(S) - f_e(A)| \leq \varepsilon W_A$.*

Theorem 2 *Protocol 3 (P3) sends $O((m+s)\log\frac{\beta N}{s})$ messages with large probability; It gets a set S of size $s = \Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ so that $|f_e(S) - f_e(A)| \leq \varepsilon W$.*

4.3.1 Sampling With Replacement

We can show similar results on s samples *with replacement*, using s independent samplers. In round j, for each element (a_n, w_n) arriving at a local site, the site generates s independent r_n values, and thus s priorities ρ_n. If any of them is larger than τ_j, then the site forwards it to coordinator, along with the index (or indices) of success.

For each of s independent samplers, say for sampler $t \in [s]$, the coordinator maintains the top 2 priorities $\rho_t^{(1)}$ and $\rho_t^{(2)}$, $\rho_t^{(1)} > \rho_t^{(2)}$, among all it received. It also keeps the element information a_t associated with $\rho^{(1)}$. For the sampler $i \in [s]$, the coordinator keeps a weight $\bar{w}_i = \rho_i^{(2)}$. One can show that $\mathbf{E}[\bar{w}_i] = W$, the total weight of the entire stream [18]. We improve the global estimate as $\hat{W} = (1/s)\sum_{i=1}^{s} \bar{w}_i$, and

then assign each element a_i the same weight $\hat{w}_i = \hat{W}/s$. Now $\mathbf{E}[\sum_{i=1}^{s} \hat{w}_i] = W$, and each a_i is an independent sample (with replacement) chosen proportional to its weight. Then setting $s = O((1/\varepsilon^2)\log(1/\varepsilon))$ it is known that these samples can be used to estimate all heavy hitters within εW with probability at least $1 - e^{-\Omega(s)}$.

The jth round terminates when the $\rho_i^{(2)}$ for all i is larger than $2\tau_j$. At this point, coordinator sets $\tau_{j+1} = 2\tau_j$, informs all sites of the new threshold and begins the $(j+1)$th round.

Communication analysis. Since this protocol is an adaptation of existing results [15], its communication is $O((m + s\log s)\log(\beta N) = O((m + \frac{1}{\varepsilon^2}\log^2\frac{1}{\varepsilon})\log(\beta N))$ messages. This result doesn't improve the error bounds or communication bounds with respect to the without replacement sampler described above, as is confirmed in Section 6. Also in terms of running time (without parallelism at each site), sampling *without replacement* will be better.

4.4 Weighted Heavy-Hitters Protocol 4

This protocol is inspired by the unweighted case from Huang *et.al.* [23]. Each site maintains an estimate of the total weight $\hat{W}$ that is provided by the coordinator and always satisfies $\hat{W} \leq W \leq 2\hat{W}$, with high probability. It then sets a probability $p = 2\sqrt{m}/(\varepsilon\hat{W})$. Now given a new element (a, w) with some probability $\bar{p}$, it sends to the coordinator $(e, \bar{w}_{e,j} = f_e(A_j))$ for $a = e \in [u]$; this is the total weight of all items in its stream that equal element e. Finally the coordinator needs to adjust each $\bar{w}_{e,j}$ by adding $1/p - 1$ (for elements that have been witnessed) since that is the expected number of items with element e in the stream until the next update for e.

If w is an integer, then one option is to pretend it is actually w distinct elements with weight 1. For each of the w elements we create a random variable Z_i that is 1 with probability p and 0 otherwise. If *any* $Z_i = 1$, then we send $f_e(A_j)$. However this is inefficient (say if $w = \beta = 1000$), and only works with integer weights.

Instead we notice that at least one Z_i is 1 if none are 0, with probability $1 - (1-p)^w \approx 1 - e^{-pw}$. So in the integer case, we can set $\bar{p} = 1 - (1-p)^w$, and then since we send a more accurate estimate of f_e (as it essentially comes later in the stream) we can directly apply the analysis from Huang *et.al.* [23]. To deal with non integer weights, we set $\bar{p} = 1 - e^{-pw}$, and describe the approach formally on a site in Algorithm 4.7.

Notice that the probability of sending an item is asymptotically the same in the case that $w = 1$, and it is smaller otherwise (since we send at most one update $\bar{w}_{e,j}$ per batch). Hence the communication bound is asymptotically the same, except for the number of rounds. Since the weight is broadcast to the sites from the coordinator whenever it doubles, and now the total weight can be βN instead of N, the number of rounds is $O(\log(\beta N))$ and the total communication is $O((\sqrt{m}/\varepsilon)\log(\beta N))$ with high probability.

Algorithm 4.7 P4: Tracking of heavy-hitters (at site S_j)

Given weight $\hat{W}$ from C, set $p = 2\sqrt{m}/(\varepsilon\hat{W})$.
for each item (a, w) it receives **do**
  For $a = e$ update $f_e(A_j) := f_e(A_j) + w$.
  Set $\bar{p} = 1 - e^{-pw}$.
  With probability $\bar{p}$ send $\bar{w}_{e,j} = f_e(A_j)$ to C.

When the coordinator is sent an estimate $\bar{w}_{e,j}$ of the total weight of element e at site j, it needs to update this estimate slightly as in Huang *et.al.*, so that it has the right expected value. It sets $\hat{w}_{e,j} = \bar{w}_{e,j} + 1/p$, where again $p = 2\sqrt{m}/(\varepsilon\hat{W})$; $\hat{w}_{e,j} = 0$ if no such messages are sent. The coordinator then estimates each $f_e(A)$ as $\hat{W}_e = \sum_{j=1}^{m} \hat{w}_e$.

We first provide intuition how the analysis works, if we used $\bar{p} = 1 - (1-p)^w$ (i.e. $\approx 1 - e^{-pw}$) and w is an integer. In this case, we can consider simulating the process with w items of weight 1; then it is identical to the unweighted algorithm, except we always send $\bar{w}_{e,j}$ at then end of the batch of w items. This means the expected number until the next update is still $1/p - 1$, and the variance of $1/p^2$ and error bounds of Huang *et.al.* [23] still hold.

Lemma 7 *The above protocol guarantees that* $|f_e(A) - \hat{W}_e| \leq \varepsilon W$ *on the coordinator, with probability at least* 0.75.

Proof. Consider a value of k large enough so that $w \cdot 10^k$ is always an integer (i.e., the precision of w in a system is at most k digits past decimal point). Then we can hypothetically simulate the unweighted case using $w_k = w \cdot 10^k$ points. Since now $\hat{W}$ represents 10^k times as many unweighted elements, we have $p_k = p/10^k = \sqrt{m}/(\varepsilon\hat{W}10^k)$. This means the probability we send an update should be $1 - (1-p_k)^{w_k}$ in this setting.

Now use that for any x that $\lim_{n\to\infty}(1 - \frac{x}{n})^n = e^{-x}$. Thus setting $n = w_k$ and $x = p_k \cdot w_k = (p/10^k)(w10^k) = pw$ we have $\lim_{k\to\infty} 1 - (1-p_k)^{w_k} = 1 - e^{-pw}$.

Next we need to see how this simulated process affects the error on the coordinator. Using results from Huang *et.al.* [23], where they send an estimate $\bar{w}_{e,j}$, the expected value $\mathbf{E}[\bar{w}_{e,j}] = f_e(A_j) - 1/p + 1$ at any point afterwards where that was the last update. This estimates the count of weight 1 objects, so in the case where they are weight 10^{-k} objects the estimate of $f_e(A_j)^{(k)} = f_e(A_j)10^k$ is using $\bar{w}_{e,j}^{(k)} = \bar{w}_{e,j}10^k$. Then, in the limiting case (as $k \to \infty$), we adjust the weights as follows.

$$\mathbf{E}[\bar{w}_{e,j}] = \mathbf{E}[\bar{w}_{e,j}^{(k)}]10^{-k} = (f_e(A_j)^{(k)} - 1/p_k + 1) \cdot 10^{-k}$$
$$= (f_e(A_j)10^k - \frac{10^k}{p} + 1)10^{-k} = f_e(A_j) - \frac{1}{p} + 10^{-k},$$

so as $\lim_{k\to\infty} \mathbf{E}[\bar{w}_{e,j}] = f_e(A_j) - 1/p$. So our procedure has the right expected value. Furthermore, it also follows that the variance is still $1/p^2$, and thus the error bound from [23] that any $|f_e(A) - \hat{W}_e| \leq \varepsilon W$ with probability at least 0.75 still holds. □

Theorem 3 *Protocol 4 (P4) sends* $O(\frac{\sqrt{m}}{\varepsilon}\log(\beta N))$ *total messages and with probability* 0.75 *has* $|f_e(A) - \hat{W}_e| \leq \varepsilon W$.

The bound can be made to hold with probability $1 - \delta$ by running $\log(2/\delta)$ copies and taking the median. The space on each site can be reduced to $O(1/\varepsilon)$ by using a weighted variant of the space-saving algorithm [32]; the space on the coordinator can be made $O(m/\varepsilon)$ by just keeping weights for which $\bar{w}_{i,e} \geq 2\varepsilon\hat{W}_j$, where $\hat{W}_j$ is a 2-approximation of the weight on site j.

5. DISTRIBUTED MATRIX TRACKING

We will next extend weighted frequent item estimation protocols to solve the problem of tracking an approximation to a distributed matrix. Each element a_n of a stream for

any site is now a row of the matrix. As we will show soon in our analysis, it will be convenient to associate a weight with each element defined as the squared norm of the row, i.e., $w_n = \|a_n\|^2$. Hence, for reasons outlined in Section 3, we assume in our analysis that the squared norm of every row is bounded by a value β. There is a designated coordinator C who has a two-way communication channel with each site and whose job is to maintain a much smaller matrix B as an approximation to A such that for any unit vector $x \in \mathbb{R}^{d\times 1}$ (with $\|x\| = 1$) we can ensure that:

$$|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2.$$

Note that the covariance of A is captured as $A^T A$ (where ^{T} represents a matrix transpose), and that the above expression is equivalent to

$$\|A^T A - B^T B\|_2 \leq \varepsilon \|A\|_F^2.$$

Thus, the approximation guarantee we preserve shows that the covariance of A is well-approximated by B. And the covariance is the critical property of a matrix that needs to be (approximately) preserved as the basis for most downstream data analysis; e.g., for PCA or LSI.

Our measures of complexity will be the communication cost and the space used at each site to process the stream. We measure communication in terms of the number of messages, where each message is a row of length d, the same as the input stream. Clearly, the space and computational cost at each site and coordinator is also important, but since we show that all proposed protocols can be run as streaming algorithms at each site, and will thus not be space or computation intensive.

Overview of protocols. The protocols for matrix tracking mirror those of weighted item frequency tracking. This starts with a similar batched streaming baseline P1. Protocol P2 again reduces the total communication bound, where a global threshold is given for each "direction" instead of the total squared Frobenious norm. Both P1 and P2 are deterministic. Then matrix tracking protocol P3 randomly selects rows with probability proportional to their squared norm and maintains an ε-sample at the coordinator. Using this sample set, we can derive a good approximation.

Given the success of protocols P1, P2, and P3, it is tempting to also extend protocol P4 for item frequency tracking in Section 4.4 to distributed matrix tracking. However, unlike the other protocols, we can show that the approach described in Algorithm 4.7 cannot be extended to matrices in any straightforward way while still maintaining the same communication advantages it has (in theory) for the weighted heavy-hitters case. Due to lack of space, we defer this explanation, and the related experimental results to the appendix section of the full version [22].

5.1 Distributed Matrix Tracking Protocol 1

We again begin with a batched version of a streaming algorithm, shown as Algorithm 5.1 and 5.2. That is we run a streaming algorithm (e.g. Frequent Directions [29], labeled FD, with error $\varepsilon' = \varepsilon/2$) on each site, and periodically send the contents of the memory to the coordinator. Again this is triggered when the total weight (in this case squared norm) has increased by $(\varepsilon/2m)W$.

As with the similar frequency tracking algorithm, based on Frequent Directions [29] satisfying the mergeable property [2], we can show this maintains at most $\varepsilon\|A\|_F^2$ total error at all times, and requires a total of $O((m/\varepsilon^2)\log(\beta N))$ total rows of communication.

Algorithm 5.1 P1: Deterministic Matrix Tracking (at S_i)

for (a_n, w_n) in round j **do**
  Update $B_i \leftarrow \mathsf{FD}_{\varepsilon'}(B_i, a_n)$; and $F_i \mathrel{+}= \|a_n\|^2$.
  if $(F_i \geq \tau = (\varepsilon/2m)\hat{F})$ **then**
    Send (B_i, F_i) to coordinator; make B_i, F_i empty.

Algorithm 5.2 P1: Deterministic Matrix Tracking (at C)

On input (B_i, F_i):
Update sketch $B \leftarrow \mathsf{Merge}_{\varepsilon'}(B, B_i)$ and $F_C \mathrel{+}= F_i$.
if $(F_C/\hat{F} > 1 + \varepsilon/2)$ **then**
  Update $\hat{F} \leftarrow F_C$, and broadcast $\hat{F}$ to all sites.

5.2 Distributed Matrix Tracking Protocol 2

Again, this protocol is based very closely on a weighted heavy-hitters protocol, this time the one from Section 4.2.

Each site S_j maintains a matrix B_j of the rows seen so far at this site and not sent to coordinator. In addition, it maintains $\hat{F}$, an estimate of $\|A\|_F^2$, and $F_j = \|B_j\|_F^2$, denoting the total squared Frobenius norm received since its last communication to C about $\hat{F}$. The coordinator C maintains a matrix B approximating A, and $\hat{F}$, an ε-approximation of $\|A\|_F^2$.

Initially each $\hat{F}$ is set to zero for all sites. When site j receives a new row, it calls Algorithm 5.3, which basically sends $\|B_j x\|^2$ in direction x when it is greater than some threshold provided by the coordinator, if one exists.

Algorithm 5.3 P2: Deterministic Matrix Tracking (at S_j)

$F_j \mathrel{+}= \|a_i\|^2$
if $(F_j \geq \frac{\varepsilon}{m}\hat{F})$ **then**
  Send F_j to coordinator; set $F_j = 0$.
Set $B_j \leftarrow [B_j; a_i]$
$[U, \Sigma, V] = \mathsf{svd}(B_j)$
for $((v_\ell, \sigma_\ell)$ such that $\sigma_\ell^2 \geq \frac{\varepsilon}{m}\hat{F})$ **do**
  Send $\sigma_\ell v_\ell$ to coordinator; set $\sigma_\ell = 0$.
$B_j = U\Sigma V^T$

On the coordinator side, it either receives a vector form message σv, or a scalar message F_j. For a scalar F_j, it adds it to $\hat{F}$. After at most m such scalar messages, it broadcasts $\hat{F}$ to all sites. For vector message $r = \sigma v$, the coordinator updates B by appending r to $B \leftarrow [B; r]$. The coordinator's protocol is summarized in Algorithm 5.4.

Lemma 8 *At all times the coordinator maintains B such that for any unit vector x*

$$\|Ax\|^2 - \varepsilon\|A\|_F^2 \leq \|Bx\|^2 \leq \|Ax\|^2 \tag{1}$$

Proof. To prove this, we also need to show it maintains another property on the total squared Frobenious norm:

$$(1 - 2\varepsilon)\|A\|_F^2 < \hat{F} \leq \|A\|_F^2. \tag{2}$$

This follows from the analysis in Section 4.2 since the squared Frobenius norm is additive, just like weights. The following analysis for the full lemma is also similar, but requires more care in dealing with matrices. First, for any x we have

$$\|Ax\|^2 = \|Bx\|^2 + \sum_{j=1}^{m} \|B_j x\|^2.$$

Algorithm 5.4 P2: Deterministic Matrix Tracking (at C)

On a scalar message F_j from site S_j
Set $\hat{F}$ += F_j and #msg += 1.
if (#msg $\geq m$) **then**
  Set #msg = 0 and broadcast $\hat{F}$ to all sites.
On a vector message $r = \sigma v$: append $B \leftarrow [B; r]$

This follows since $\|Ax\|^2 = \sum_{i=1}^n \langle a_i, x\rangle^2$, so if nothing is sent to the coordinator, the sum can be decomposed like this with B empty. We just need to show the sum is preserved when a message $r = \sigma_1 v_1$ is sent. Because of the orthogonal decomposition of B_j by the svd$(B_j) = [U, \Sigma, V]$, then $\|B_j x\|^2 = \sum_{\ell=1}^d \langle \sigma_\ell v_\ell, x\rangle^2$. Thus if we send any $\sigma_\ell v_\ell$ to the coordinator, append it to B, and remove it from B_j, the sum is also preserved. Thus, since the norm on B is always less than on A, the right side of (1) is proven.

To see the left side of (1) we need to use (2), and show that not too much mass remains on the sites. First we bound $\|B_j x\|^2$.

$$\|B_j x\|^2 = \sum_{\ell=1}^d \sigma_\ell^2 \langle v_\ell, x\rangle^2 \leq \sum_{\ell=1}^d \frac{\varepsilon}{m}\hat{F}\langle v_\ell, x\rangle^2 = \frac{\varepsilon}{m}\hat{F} \leq \frac{\varepsilon}{m}\|A\|_F^2.$$

And thus $\sum_{j=1}^m \|B_j x\|^2 \leq m\frac{\varepsilon}{m}\|A\|_F^2 = \varepsilon\|A\|_F^2$ and hence

$$\|Ax\|^2 \leq \|Bx\|^2 + \sum_{j=1}^m \|B_j x\|^2 \leq \|Bx\|^2 + \varepsilon\|A\|_F^2. \quad \square$$

The communication bound follows directly from the analysis of the weighted heavy hitters since the protocols for sending messages and starting new rounds are identical with $\|A\|_F^2$ in place of W, and with the squared norm change along the largest direction (the top right singular value) replacing the weight change for a single element. Thus the total communication is $O(\frac{m}{\varepsilon}\log(\beta N))$.

Theorem 4 *For a distributed matrix A whose squared norm of rows are bounded by β and for any $0 \leq \varepsilon \leq 1$, the above protocol (P2) continuously maintains $\hat{A}$ such that $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \varepsilon\|A\|_F^2$ and incurs a total communication cost of $O((m/\varepsilon)\log(\beta N))$ messages.*

Bounding space at sites. It is possible to also run a small space streaming algorithm on each site j, and also maintain the same guarantees. The Frequent Directions algorithm [29], presented a stream of rows a_i forming a matrix A, maintains a matrix $\tilde{A}$ using $O(1/\varepsilon')$ rows such that $0 \leq \|Ax\|^2 - \|\tilde{A}x\|^2 \leq \varepsilon'\|A\|_F^2$ for any unit vector x.

In our setting we run this on two matrices on each site with $\varepsilon' = \varepsilon/4m$. (It can actually just be run on B_j, but then the proof is much less self-contained.) It is run on A_j, the full matrix. Then instead of maintaining B_j that is A_j after subtracting all rows sent to the coordinator, we maintain a second matrix S_j that contains all rows sent to the coordinator; it appends them one by one, just as in a stream. Now $\|B_j x\|^2 = \|A_j x\|^2 - \|S_j x\|^2$. Thus if we replace both A_j with $\tilde{A}_j$ and S_j with $\tilde{S}_j$, then we have

$$\|B_j x\|^2 = \|A_j x\|^2 - \|S_j x\|^2 \leq \|\tilde{A}_j x\|^2 - \|\tilde{S}_j x\|^2 + \frac{\varepsilon}{4m}\|A_j\|_F^2,$$

and similarly $\|B_j x\|^2 \geq \|\tilde{A}_j x\|^2 - \|\tilde{S}_j x\|^2 - \frac{\varepsilon}{4m}\|A_j\|_F^2$ (since $\|S_j\|_F^2 \leq \|A_j\|_F^2$). From here we will abuse notation and write $\|\tilde{B}_j x\|^2$ to represent $\|\tilde{A}_j x\|^2 - \|\tilde{S}_j x\|^2$.

Now we send the top singular vectors v_ℓ of $\tilde{B}_j$ to the coordinator only if $\|\tilde{B}_j v_\ell\|^2 \geq \frac{3\varepsilon}{4m}\hat{F}$. Using our derivation, thus we only send a message if $\|B_j v_\ell\|^2 \geq \frac{\varepsilon}{2m}\|A\|_F^2$, so it only sends at most twice as many as the original algorithm. Also if $\|B_j v_\ell\|^2 > \frac{\varepsilon}{m}\|A\|_F^2$ we always send a message, so we do not violate the requirements of the error bound.

The space requirement per site is then $O(1/\varepsilon') = O(m/\varepsilon)$ rows. This also means, as with Frequent Directions [29], we can run Algorithm 5.3 in batch mode, and only call the svd operation once every $O(1/\varepsilon')$ rows.

It is straightforward to see the coordinator can also use Frequent Directions to maintain an approximate sketch, and only keep $O(1/\varepsilon)$ rows.

5.3 Distributed Matrix Tracking Protocol 3

Our next approach is very similar to that discussed in Section 4.3. On each site we run Algorithm 4.5, the only difference is that for an incoming row a_i, it treats it as an element $(a_i, w_i = \|a_i\|^2)$. The coordinator's communication pattern is also the same as Algorithm 4.6, the only difference is how it interprets the data it receives.

As such, the communication bound follows directly from Section 4.3; we need $O((m+(1/\varepsilon^2)\log(1/\varepsilon))\log(\beta N\varepsilon))$ messages, and we obtain a set S of at least $s = \Theta((1/\varepsilon)^2\log(1/\varepsilon))$ rows chosen proportional to their squared norms; however if the squared norm is large enough, then it is in the set S deterministically. To simplify notation we will say that there are exactly s rows in S.

Estimation by coordinator. The coordinator "stacks" the set of rows $\{a_1, \ldots, a_s\}$ to create an estimate $B = [a_1; \ldots; a_s]$. We will show that for any unit vector x that $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon\|A\|_F^2$.

If we had instead used the weighted sampling with replacement protocol from Section 4.3.1, and retrieved $s = O(1/\varepsilon^2)$ rows of A onto the coordinator (sampled proportionally to $\|a_i\|^2$ and then rescaled to have the same weight), we could immediately show the desired bound was achieved using know results on column sampling [17]. However, as is the case with weighted heavy-hitters, we can achieve the same error bound for without replacement sampling in our protocol, and this uses less communication and running time.

Recall for rows a_i such that $\|a_i\|^2 \geq \hat{\rho}$, (for a priority $\hat{\rho} < 2\tau$) it keeps them as is; for other rows, it rescales them so their squared norm is $\hat{\rho}$. And $\hat{\rho}$ is defined so that $\mathbf{E}[\|B\|_F^2] = \|A\|_F^2$, thus $\hat{\rho} \leq W/s$.

Theorem 5 *Protocol 3 (P3) uses $O((m+s)\log(\beta N/s))$ messages of communication, with $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$, and for any unit vector x we have $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon\|A\|_F^2$, with probability at least $1 - 1/s$.*

Proof. The error bound roughly follows that of Lemma 6. We apply the same negatively correlated Chernoff-Hoeffding bound but instead define random variable $X_{i,x} = \langle a_i, x\rangle^2$. Thus $M_x = \sum_{i=1}^s X_{i,x} = \|Bx\|^2$. Again $\Delta = \hat{\rho}$ (since elements with $\|a_i\|^2 > \hat{\rho}$ are not random) and $\mathbf{E}[M_x] = \|Ax\|^2$. It again follows that

$$\mathbf{Pr}[|\|Bx\|^2 - \|Ax\|^2| \leq \varepsilon\|A\|_F^2/2] \leq \exp(-\varepsilon^2 s/32) \leq \delta.$$

Setting $\delta = \Omega(1/s)$ yields that when $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$ this holds with probability at least $1 - \delta = 1 - 1/s = 1 - 1/\Theta((1/\varepsilon)^2\log(1/\varepsilon))$, for any unit vector x. $\square$

We need $O(1)$ space per site and $O(s)$ space on coordinator.

6. EXPERIMENTS

Datasets. For tracking the distributed weighted heavy hitters, we generated data from Zipfian distribution, and set the skew parameter to 2 in order to get meaningful distributions that produce some heavy hitters per run. The generated dataset contained 10^7 points, in order to assign them weights we fixed the upper bound (default $\beta = 1,000$) and assigned each point a uniform random weight in range $[1, \beta]$. Weights are not necessarily integers.

For the distributed matrix tracking problem, we used two large real datasets "PAMAP" and "YearPredictionMSD", from the machine learning repository of UCI.

PAMAP is a Physical Activity Monitoring dataset and contains 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 $N = 629,250$ rows and $d = 44$ columns (removing columns containing missing values), giving a $N \times d$ matrix (when running to the end). This matrix is low-rank.

YearPredictionMSD is a subset from the "Million Songs Dataset" [6] and contains the prediction of the release year of songs from their audio features. It has over 500,000 rows and $d = 90$ columns. We used a subset with $N = 300,000$ rows, representing a $N \times d$ matrix (when running to the end). This matrix has high rank.

Metrics. The efficiency and accuracy of the weighted heavy hitters protocols are controlled with input parameter ε specifying desired error tolerance. We compare them on:

- *Recall:* The number of true heavy hitters returned by a protocol over the correct number of true heavy hitters.
- *Precision:* The number of true heavy hitters returned by a protocol over the total number of heavy hitters returned by the protocol.
- err: Average relative error of the frequencies of the true heavy hitters returned by a protocol.
- msg: Number of messages sent during a protocol.

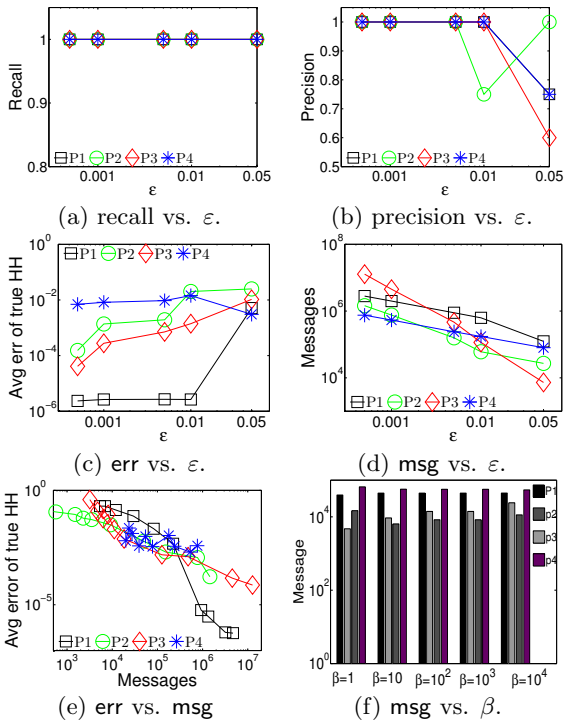
For matrix approximation protocols, we used:

- err: Defined as $\|A^TA - B^TB\|_2/\|A\|_F^2$, where A is the input matrix and B is the constructed low rank approximation to A. It is equivalent to the following: $\max_{\{x,\ \|x\|=1\}}(\|Ax\|^2 - \|Bx\|^2)/\|A\|_F^2$.
- msg: Number of messages (scalar-form and vector-form) sent during a protocol.

We observed that both the approximation errors and communication costs of all methods *are very stable with respect to query time*, by executing estimations at the coordinator at randomly selected time instances. Hence, we only report the average err from queries in the very end of the stream (i.e., results of our methods on really large streams).

6.1 Distributed Weighted Heavy Hitters

We denote four protocols for tracking distributed weighted heavy hitters as P1, P2, P3 and P4 respectively. As a baseline, we could send all 10^7 stream elements to the coordinator, this would have no error. All of our heavy hitters protocols return an element e as heavy hitter only if $\hat{W}_e/\hat{W} \geq \phi - \varepsilon/2$ while the exact weighted heavy hitter



(a) recall vs. ε. (b) precision vs. ε.
(c) err vs. ε. (d) msg vs. ε.
(e) err vs. msg (f) msg vs. β.

Figure 1: Results for distributed weighted heavy hitters protocols on Zipfian distribution with skew=2.

method which our protocols are compared against, returns e as heavy hitter if $f_e(A)/W \geq \phi$.

We set the heavy-hitter threshold ϕ to 0.05 and we varied error guarantee ε in the range $\{5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$. When the plots do not vary ε, we use the default value of $\varepsilon = 10^{-3}$. Also we varied number of sites (m) from 10 to 100, otherwise we have as default $m = 50$.

All four algorithms prove to be highly effective in estimating weighted heavy hitters accurately, as shown in *recall* (Figure 1(a)) and *precision* (Figure 1(b)) plots. In particular, the recall values for all algorithms are constant 1.0.

Note that precision values dip, but this is because the true heavy hitters have $f_e(A)/W$ above ϕ where our algorithms only return a value if $\hat{W}_e/\hat{W} \geq \phi - \varepsilon/2$, so they return more false positives as ε increases. For ε smaller than 0.01, all protocols have a precision of 1.0.

When measuring (the measured) err as seen in Figure 1(c), our protocols consistently outperform the error parameter ε. The only exception is P4, which has slightly larger error than predicted for very small ε; recall this algorithm is randomized and has a constant probability of failure. P1 has almost no error for $\varepsilon = 0.01$ and below; this can be explained by improved analysis for Misra-Gries [5] on skewed data, which applies to our Zipfian data. Protocols P2 and P3 also greatly underperform their guaranteed error.

The protocols are quite communication efficient, saving several orders of magnitude in communication as shown in Figure 1(d). For instance, all protocols use roughly 10^5 messages at $\varepsilon = 0.01$ out of 10^7 total stream elements. To further understand different protocols, we tried to compare them by making them using (roughly) the same number of messages. This is achieved by using *different* ε values. As shown in Figure 1(e); all protocols achieved excellent approximation quality, and the measured error drops quickly

DataSet	PAMAP, $k = 30$		MSD, $k = 50$	
Method	err	msg	err	msg
P1	7.5859e-06	628537	0.0057	300195
P2	0.0265	10178	0.0695	6362
P3WOR	0.0057	3962	0.0189	3181
P3WR	0.0323	25555	0.0255	22964
FD	2.1207e-004	629250	0.0976	300000
SVD	1.9552e-006	629250	0.0057	300000

Table 1: Raw numbers of PAMAP and MSD.

as we allocate more budget for the number of messages. In particular, P2 is the best if fewer than 10^5 messages are acceptable with P3 also shown to be quite effective. P1 performs best if 10^6 messages are acceptable.

In another experiment, we tuned all protocols to obtain (roughly) the same measured error of err $= 0.1$ to compare their communication cost versus the upper bound on the element weights (β). Figure 1(f) shows that they are all robust to the parameter β; P2 or P3 performs the best.

6.2 Distributed Matrix Tracking

The strong results for distributed weighted heavy hitter protocols transfer over empirically to the results for distributed matrix tracking, but with slightly different tradeoffs. Again, we denote our three protocols by P1, P2, and P3 in all plots. *As a baseline*, we consider two algorithms: they both send all data to the coordinator. One calls Frequent-Directions (FD) [29], and second calls SVD which is optimal but not streaming. In all remaining experiments, we have used default value $\varepsilon = 0.1$ and $m = 50$, unless specified. Otherwise ε varied in range $\{5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}, 5 \times 10^{-1}\}$, and m varied in range $[10, 100]$.

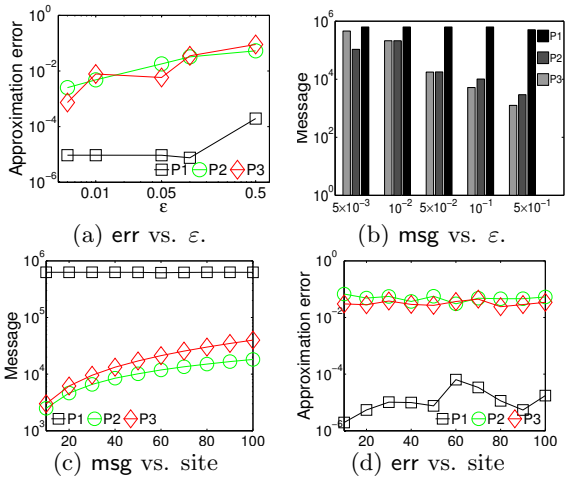
Table 1 compares all algorithms, including SVD and FD to compute rank k approximations of the matrices, with $k = 30$ and $k = 50$ on PAMAP and MSD respectively. Since err values for the two offline algorithms are minuscule for PAMAP, it indicates it is a low rank matrix (less than 30), where as MSD is high rank, since error remains, even with the best rank 50 approximation from the SVD method.

Note that P3WOR and P3WR refer to Protocol 3, *without replacement* and *with replacement* sampling strategies, respectively. As predicted by the theoretical analysis, we see that P3WOR outperforms P3WR in both settings, always having much less error and many fewer messages. Moreover, P3WOR will gracefully shift to sending all data deterministically with no error as ε becomes very small. Hence we only use P3WOR elsewhere, labeled as just P3.

Also note that P1 in the matrix scenario is far less effective; although it achieves very small error, it sends as many messages (or more) as the naive algorithms. Little compression is taking place by FD at distributed sites before the squared norm threshold is reached.

Figures 2(a) and 3(a) show as ε increases, error of protocols increases too. In case of P3 this observation is justified by the fact P3 samples $O((1/\varepsilon^2)\log(1/\varepsilon))$ elements, and as ε increases, it samples fewer elements, hence results in a weaker estimation of true heavy directions. In case of P2, as ε increases, they allocate a larger error slack to each site and sites communicate less with the coordinator, leading to a coarse estimation. Note that again P1 vastly outperforms its error guarantees, this time likely explained via the improved analysis of Frequent-Directions [21].

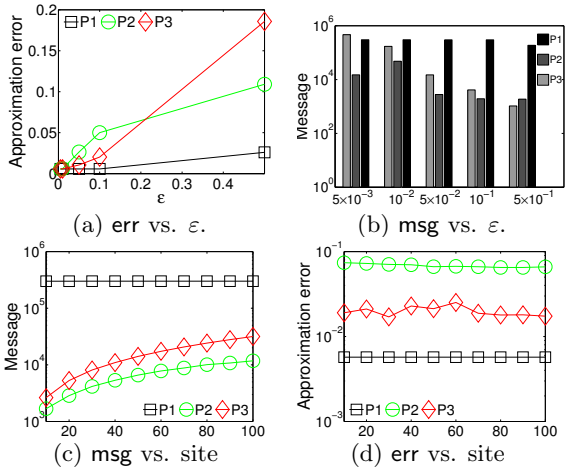
Figures 2(b) and 3(b) show number of messages of each



(a) err vs. ε. (b) msg vs. ε.

(c) msg vs. site (d) err vs. site

Figure 2: Experiments for PAMAP dataset



(a) err vs. ε. (b) msg vs. ε.

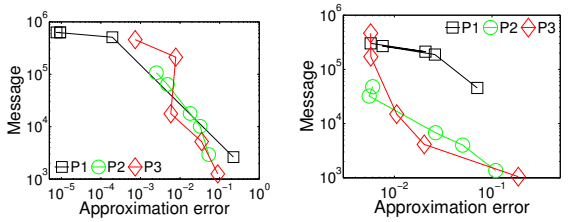
(c) msg vs. site (d) err vs. site

Figure 3: Experiments for MSD dataset

protocol vs. error guarantee ε. As we see, in large values of ε (say for $\varepsilon > 1/m = 0.02$), P2 typically uses slightly more messages than P3. But as ε decreases, P3 surpasses P2 in number of messages. This confirms the dependency of their asymptotic bound on ε ($1/\varepsilon^2$ vs. $1/\varepsilon$). P1 generally sends much more messages than both P2 and P3.

Next, we examined the number of sites (m). Figures 2(c) and 3(c) show that P2 and P3 used more communication as m increases, showing a linear trend with respect to m. P1 shows no trend since its communication depends solely on the total weight of the stream. Note that P1 sends its whole sketch, hence fix number of messages, whenever it reaches the threshold. As expected, the number of sites does not have significant impact on the measured approximation error in any protocol; see Figures 2(d) and 3(d).

We also compared the performance of protocols by tuning the ε parameter to achieve (roughly) the same measured error. Figure 4 shows their communication cost (#msg) vs the err. As shown, protocols P1, P2, and P3 incur less error with more communication and each works better in various regimes of the err versus msg trade-off. P1 works the best when the smallest error is required, but more communication is permitted. Even though its communication is the same as the naive algorithms in these examples, it allows each site and the coordinator to run small space algorithms. For smaller communication requirements (several of orders



(a) PAMAP (b) MSD

Figure 4: Comparing the two protocols: msg vs. err

of magnitude smaller than the naive methods), then either P2 or P3 are recommended. P2 is deterministic, but P3 is slightly easier to implement. Note that since MSD is high rank, and even the naive SVD or FD do not achieve really small error (e.g. 10^{-3}), it is not surprising that our algorithms do not either.

7. CONCLUSION

We provide the first protocols for monitoring weighted heavy hitters and matrices in a distributed stream. They are backed by theoretical bounds and large-scale experiments. Our results are based on important connections we establish between the two problems. Interesting open problems include, but are not limited to, extending our results to the sliding window model, and investigating distributed matrices that are column-wise distributed (i.e., each site reports values from a fixed column in a matrix).

8. REFERENCES

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *STOC*, 2001.

[2] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In *PODS*, 2012.

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

[4] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[5] R. Berinde, G. Cormode, P. Indyk, and M. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4), 2010.

[6] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR*, 2011.

[7] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20 – 30, 2006.

[8] K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *STOC*, 2009.

[9] J. C. Corbett and et al. Spanner: Google's globally distributed database. *ACM TOCS*, 31(3), 2013.

[10] G. Cormode. The continuous distributed monitoring model. *SIGMOD Record*, 42, 2013.

[11] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[12] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[13] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[14] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. *TALG*, 7(2):21, 2011.

[15] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *JACM*, 59(2), 2012.

[16] P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. In *SODA*, 2003.

[17] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SICOMP*, 36(1):158–183, 2006.

[18] N. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *JACM*, 54(32), 2007.

[19] A. Frieze, R. Kannan, and S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *JACM*, 51(6):1025–1041, 2004.

[20] R. Fuller and M. Kantardzic. FIDS: Monitoring frequent items over distributed data streams. In *MLDM*. 2007.

[21] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximation. In *SODA*, 2014.

[22] M. Ghashami, J. M. Phillips, and F. Li. Continuous matrix approximation on distributed data. ArXiv:1404.7571, 2014. `http://arxiv.org/abs/1404.7571`.

[23] Z. Huang, K. Yi, and Q. Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *PODS*, 2012.

[24] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD*, 2001.

[25] R. Y. Hung and H.-F. Ting. Finding heavy hitters over the sliding window of a weighted data stream. In *LATIN*. 2008.

[26] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, 2006.

[27] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, 2004.

[28] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *SIGMETRICS*, 2004.

[29] E. Liberty. Simple and deterministic matrix sketching. In *SIGKDD*, 2013.

[30] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *PNAS*, 104:20167–20172, 2007.

[31] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

[32] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM TODS*, 31(3):1095–1133, 2006.

[33] J. Misra and D. Gries. Finding repeated elements. *Sc. Comp. Prog.*, 2:143–152, 1982.

[34] I. Mitliagkas, C. Caramanis, and P. Jain. Memory limited, streaming pca. In *NIPS*, 2013.

[35] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[36] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SICOMP*, 26:350–368, 1997.

[37] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, 2005.

[38] K. Papagiannaki, N. Taft, and A. Lakhina. A distributed approach to measure ip traffic matrices. In *SIGCOMM IMC*, 2004.

[39] Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, and Z. Zhang. MadLINQ: large-scale distributed matrix computation for the cloud. In *EuroSys*, 2012.

[40] V. Strumpen, H. Hoffmann, and A. Agarwal. A stream algorithm for the svd. Technical report, MIT, 2003.

[41] M. Szegedy. The DLT priority sampling is essentially optimal. *STOC*, 2006.

[42] S. Tirthapura and D. P. Woodruff. Optimal random sampling in distributed streams revisited. In *PODC*, 2011.

[43] A. C.-C. Yao. Some complexity questions related to distributive computing. In *STOC*, 1979.

[44] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.