Distributing Frequency-Dependent Data Stream Computations *

Sumit Ganguly †

Abstract

The data streaming model of computation processes a sequence of continuously arriving data in a single-pass over the input using sub-linear space. For efficiency purposes, it is often desirable to perform the computations in a highly distributed fashion, as are currently done in internet applications and sensor networks. The distributed computation is typically performed using a tree-topology, where the nodes are processing elements and the data resides in the leaves of the tree. A large class of interesting and practical functions are symmetric functions (i.e., invariant of the permutation of data at the leaves) of the inputs or their approximations. Flexible distributed processing refers to the class of distributed tree computations whose output remains within a pre-specified tolerance limit regardless of the permutation of the data on the leaves or the topology of the computation tree.

We consider the question: Do efficient flexible distributed computations exist corresponding to data streaming algorithms for computing the same function/approximation? The work by [2] shows that corresponding to any deterministic algorithm that computes a total symmetric function of an input stream, there exists a flexible distributed algorithm for computing the same function of the concatenation of the input streams at the leaves of the tree from left to right. Further, they present resource bounds of the distributed algorithm in terms of the resource bounds of the streaming algorithm. In this paper, we show the existence of efficient flexible distributed algorithms corresponding to a given data stream algorithm that computes an approximation to the function of the frequency vector of the stream. As opposed to MUD, the data stream algorithm need not compute a symmetric function of the input stream.

*Preliminary version of this paper appeared in CATS: Computing, The Australasian Theory Symposium 2009, Wellington, New Zealand. Conferences in Research and Practice, CRPIT Vol. 94, Rod Downey and Prabhu Manyem, Eds.

†Affiliation: Indian Institute of Technology, Kanpur, India. E-mail: sganguly@cse.iitk.ac.in

1 Introduction

Modern distributed data-centric applications typically process massive amounts of data over a collection of nodes that are organized in the form of a tree. The applications include internet data processing (e.g., the Google MapReduce framework of scalable and flexible distributed processing as presented by [1]) and sensor networks. In this model, the distributed computation is modeled as a tree, where data resides at the leaf nodes of a (typically, depth one) tree. Data at each of the nodes is locally reduced, and the reduced data is sent to the root site, which combines the locally reduced copies into a single copy and then computes an answer from it. Finally, the relevant function is computed at the root of the tree.

Another model that has recently emerged for data-centric applications is the data streaming model. A data stream is modeled as a sequence with elements from a finite alphabet Σ. The input stream is processed in an online fashion, typically by constructing a summary in a space economical data structure. In many data streaming applications, the volume of data can be very large, even potentially infinite (e.g., network switches, sensors etc. as sources of data), and the arrival rates can be very high. Therefore, it is important for the data stream processing system to be efficient in terms of space used by the summary structure and the time required for processing each arriving stream element.

The general question that we study in this paper is: given a data streaming algorithm that computes an approximation to a function of the input stream, is it possible to obtain a flexible distributed computation of the same approximation to the function, where the sequential input stream may now be viewed as being segmented and distributed among the leaf nodes of the distributed computation tree?

This connection was considered via a model for flexible distributed processing named MUD (Massive Unordered Distributed algorithms) presented by [2]. Let n be the domain of items in the input stream and let m be the size of a stream (number of records). Their main result is: every deterministic data streaming algorithm that computes a symmetric function $g(\sigma)$ of its input stream σ of size m using time $\textsc{Time}(n, m)$, space $\textsc{space}(n, m)$ and communication $\textsc{Comm}(n, m)$ can be transformed into a flexible distributed algorithm that computes the same function g on the concatenation of the streams input to the leaves of the tree from left to right, using communication $\textsc{Comm}(n, m)$, space $O((\textsc{space}(n, m))^2)$ and time $n \times 2^{O((\textsc{space}(n,m))^2)}$.

In this work, we extend the above result as follows. We show that given any deterministic algorithm that computes some *approximation* to a func-

tion g of the *frequency vector* of its input stream using communication $\textsc{Comm}(n, m)$, there exists a flexible distributed algorithm that computes the same approximation to the function g on the sum of the frequency vectors of the streams input to the leaves of the tree, using communication $\textsc{Comm}(n, m)$ and time $O(n \times \textsc{Comm}(n, m))$.

The rest of the paper is organized as follows. In Section 2, we present a model for flexible distributed processing called FDP. Section 3 presents an overview of data streaming algorithms and recalls the main result of [2]. In Section 4, we present our result.

2 A Model for Distributed Processing

In this section, we describe a model of flexible distributed computations referred to in the previous section called the FDP model (Flexible Distributed Processing).

An FDP distributed computation is defined by four components, namely, a computation tree T, the reduction function ϕ, the message composition function $\odot$ and the output function ψ. The nodes of the computation tree are processing elements. The input data resides in the leaf nodes of T. Since we are interested in distributed streaming applications, we will assume that the input datum at each of the leaf nodes is a data stream, that is, an element of Σ^*.

The computation proceeds bottom-up in the tree as follows. Each leaf node $v_l \in T$ applies a reduction function (data summary/synposis function) $\phi : \Sigma^* \to M$ to its input stream σ_l to obtain a message $\phi(\sigma_l)$. This message is then sent to its parent. The computation is homogeneous, that is, all leaf nodes apply the same reduction function ϕ to their respective input streams.

Each internal node applies the message composition function $\odot : M \times M \times \ldots \times M \to M$ to combine the messages received from its children, ordered from left to right, to produce a new summary message that is then conveyed to its parent. Consider an internal node v with k children nodes ordered $u_1, u_2, \ldots, u_k$ from left to right. If m_j is the message sent by node u_j, then the node v composes its output message as

$$\odot(m_1, m_2, \ldots, m_k) \ .$$

The message composition function in general may not be symmetric, although, for all the classes considered and reviewed in this paper, this is indeed the case.

Let

$$m_{\text{root}} = \odot(T(\sigma_1, \ldots, \sigma_k))$$

denote the data summary message obtained at the root of the computation tree T that has k leaves and with input streams $\sigma_1, \ldots, \sigma_k$ at the leaves ordered from left to right. The root of the computation tree T produces the output

$$\psi(m_{\text{root}})$$

where, $\psi : M \rightarrow O$ processes a data summary message and returns an output.

The distributed computation is fully specified by the four-tuple $P = (T, \phi, \odot, \psi)$. If T has k leaves and the sequence of input streams in the left to right ordering of the leaves is $\sigma_1, \ldots, \sigma_k$, then the output of the tree for this leaf input sequence is denoted by

$$P(\sigma_1, \ldots, \sigma_k) = \psi(\odot(T(\sigma_1, \ldots, \sigma_k))) \ .$$

An FDP algorithm template, or simply an FDP algorithm, is denoted by the triple $(\phi, \odot, \psi)$. It defines a distributed computation together with a computation tree T and input streams at the leaves of the tree.

Notation: Total and symmetric functions. A function $g : \Sigma^* \rightarrow O$ will be said to be *total* if it is defined for all sequences in Σ^*. A function $g : \Sigma^* \rightarrow O$ is said to be symmetric if $g(\sigma) = g(\pi(\sigma))$ for any permutation π over σ, and any $\sigma \in \Sigma^*$.

Approximate Computation. A notion of approximation is specified by a binary approximation predicate APPROX : $O \times O \rightarrow \{\text{TRUE}, \text{FALSE}\}$ such that APPROX$(\hat{o}, o)$ returns TRUE if $\hat{o}$ is an acceptable approximation to the exact value o and is FALSE otherwise. This notion of approximation subsumes definitions that are based on closeness of distances, etc.. We note that approximations to symmetric functions need not be symmetric.

Given two streams σ and τ, we denote the concatenation of σ followed by τ as the stream $\sigma \circ \tau$. Given a subset $F \subset \Sigma^*$ representing the set of feasible input streams, an FDP algorithm $(\phi, \odot, \psi)$ is said to *compute a function* $g : F \rightarrow O$ *approximately* with respect to the approximation predicate APPROX provided the following holds.

For every input stream $\sigma \in F$, for any number k of partitions of the input stream, any partition of σ into k contiguous segments $\sigma = \sigma_1 \odot \sigma_2 \odot \ldots \odot \sigma_k$, any permutation π over $\{1, 2, \ldots, k\}$ and any computation tree T with k leaves with inputs $\sigma_{\pi(1)}, \ldots, \sigma_{\pi(k)}$ at the leaves ordered from left to right,

the output of the distributed computation $P = (T, \phi, \odot, \psi)$ approximates $g(\sigma)$ with respect to the approximation predicate APPROX. That is,

$$\text{APPROX}(P(T(\sigma_{\pi(1)}, \ldots, \sigma_{\pi(k)})), g(\sigma)) = \text{TRUE} \ .$$

The space and time requirements of an FDP algorithm $(\phi, \odot, \psi)$ are measured as the maximum of the respective requirements for the functions ϕ and $\odot$ respectively (the output function ψ is excluded), as a function of the alphabet size $n = |\Sigma|$ and m, the number of elements in the sequence. The communication is measured as the maximum number of bits sent from a child node to its parent, as a function of the input stream size.

The FDP[VECSUM] is a sub-class of FDP algorithms where each leaf node v_l has an n-dimensional integer vector f_l. The algorithm computes an approximation to some function $g : \mathbb{Z}^n \to O$ defined as the application of g to the sum of the distributed vectors, that is,

$$g\Big(\sum_{\text{leaves } l} f_l \Big) \ .$$

The class FDP[VECSUM] is very interesting in practice. For example, in a large network, there may be a collection of k distinct traffic matrix monitoring centers, corresponding to nodes. It is needed to aggregate the data by adding the matrices (or, vectors, tensors, etc.) coordinate-wise, and then, performing some analysis on the resulting "global" matrix. Examples of functions include, statistical functions such as median and quantiles, moments of frequency, items with high aggregate frequencies, etc..

The MUD model [2]

A MUD algorithm is a deterministic FDP algorithm $(\phi, \odot, \psi)$ that satisfies the following properties.

1. The FDP algorithm is total, that is, the feasible set of input streams $F = \Sigma^*$.

2. For any given input stream σ and an arbitrary partition of σ into any number $k \geq 1$ sub-streams $\sigma_1, \ldots, \sigma_k$ and any computation tree T with input data on the leaves being $\sigma_1, \ldots, \sigma_k$, and any permutation π over $\{1, 2, \ldots, k\}$, the output of the program $P(T(\sigma_{\pi(1)}, \ldots, \sigma_{\pi(k)}))$ is invariant of k, T and π.

MUD computations form a special case of FDP, namely, (a) the function computed by all trees is the same, and, (b) the computed function is a symmetric function of the concatenation of the input streams appearing at the leaves. FDP computations do not require that the answers be independent of the trees or orderings of the input–in fact, the answers may be different. We will present the main theorem of MUD after formally defining stream computations.

3 Stream Automaton

We model a general stream over the domain $[n] = \{1, 2, \ldots, n\}$ as a sequence of individual records of the form $(index, v)$, where, *index* represents the position of this record in the sequence and v belongs to the set $\Sigma = \Sigma_n = \{e_1, -e_1, \ldots, e_n, -e_n\}$. Here, e_i refers to the n-dimensional elementary vector $(0, \ldots, 0, 1\ (i\text{th position}), 0 \ldots, 0)$. The *frequency* of a data stream σ, denoted by $f(\sigma)$ is defined as the sum of the elementary vectors in the sequence. That is,

$$f(\sigma) = \sum_{(index,v)\in\sigma} v \ .$$

A popular class of data processing applications over data streams can be modeled as exact or approximate computations over the frequency vector of the stream. Examples of such functions include, finding frequent items in a data stream, finding the median/quantiles, approximating the frequency vector by a histogram, computing the ℓ_p norms of the frequency vector, etc.. This class of functions are called *frequency-dependent* functions and denoted as STR[FREQ].

The concatenation of two streams σ and τ is denoted by $\sigma \circ \tau$. The size of a data stream σ is defined as follows.

$$|\sigma| = \max_{\sigma' \text{ sub-sequence of } \sigma} \|f(\sigma')\|_\infty \ . \tag{1}$$

The work by [3] defines the size of a stream as $\max_{\sigma' \text{ prefix of } \sigma} \|f(\sigma')\|$. The use of sub-sequence instead of prefix makes the definition slightly better.

A deterministic **stream automaton** is an abstraction for deterministic algorithms for processing data streams. It is defined as a two tape Turing machine, where the first tape is a one-way (unidirectional) input tape that contains the sequence σ of updates that constitute the stream. Each update is a member of Σ, that is, it is an elementary vector or its inverse, e_i or $-e_i$.

The second tape is a (bidirectional) two way work-tape. A configuration of a stream automaton is modelled as a triple (q, h, w), where, q is a state of the finite control, h is the current head position of the work-tape and w is the content of the work-tape. The set of configurations of a stream automaton is generally not finite; the finite description is given by the two-tape Turing machine described earlier. A stream automaton may be described as a five-tuple $A = (n, C, o, \oplus, \psi)$, where,

1. n indicates that the domain is the set $[n]$,
2. C is the set of configurations that the Turing machine can be in, after completely processing an initial prefix of some input stream. The special configuration $o \in C$ is the initial configuration. The function $\oplus : C \times \Sigma \rightarrow C$ is the transition function that takes the current configuration a and the current stream record v and returns the next configuration, $\oplus(a, v)$ and,
3. $\psi : C \rightarrow O$ is the output function.

We write the transition function in infix notation as $a \oplus v$, where, $a \in C$ and $v \in \Sigma$. We also generalize the notation so that $a \oplus \sigma$ denotes the current configuration of the automaton starting from configuration a and processing the records of the stream σ in sequence. Since the online processing proceeds from left to right, we write

$$a \oplus (\sigma \circ \tau) \stackrel{\text{def}}{=} (a \oplus \sigma) \oplus \tau \ .$$

A stream automaton A processes its input stream sequence σ in a one-way fashion from left to right. After processing all the records of the input, the stream automaton prints the output

$$\psi(o \oplus \sigma) \ .$$

The automaton $A = A_n$ is said to have a space function $\text{SPACE}(n, m)$ provided, for all input streams σ such that $|\sigma| \leq m$, the number of cells used on the worktape during the processing of input is bounded above by $\text{SPACE}(n, m)$. The space function does not include the space used by the automaton A to print its output. This allows the automaton to print an output of size much larger than $\text{SPACE}(n, m)$.

The set $C_m(A)$ represents the finite set of configurations that are reachable from the initial configuration o after completely processing an input

stream whose size is m or less. The communication function of a stream automaton A over the domain of items $[n]$ is defined as

$$\textsc{Comm}(A, n, m) = \log\left|C_m(A)\right| \ .$$

The online processing time function of an automaton A, denoted by $\textsc{Time}(n, m)$, is the time complexity of the mapping $\oplus$.

The basic result of [2] is the following.

Theorem 1 ([2])**.** *For every $s(n, m)$ space, $t(n, m)$ time and $c(n, m)$ communication streaming algorithm that computes a symmetric function $f : \Sigma^m \to O$, with $|\Sigma| = n$, $s(n, m) = \Omega(\log m)$ and $t(n, m) = \Omega(\log m)$, there exists a* MUD *algorithm that uses communication $O(c(n, m))$, space $O((s(n, m))^2)$ and time $n \times 2^{O(s(n,m))^2}$).*

The work by [2] also shows that if any of the premises of the above theorem is violated, then there exist functions and data streaming algorithms for computing them such that there does not exist a corresponding MUD algorithm with the stated properties.

We state the main property of stream automata.

Theorem 2 (by [3])**.** . *For every stream automaton $A = (n, C_A, o_A, \oplus_A, \psi_A)$, there exists a stream automaton $B = (n, C_B, o_B, \oplus_B, \psi_B)$ such that the following holds.*

(1.) *For any* APPROX *predicate and any total function $g : \mathbb{Z}^n \to O$,* $\textsc{Approx}(\psi_B(\sigma), g(\sigma))$ *holds if* $\textsc{Approx}(\psi_A(\sigma), g(\sigma))$ *holds.*

(2.) $\textsc{Comm}(B, n, m) \leq \textsc{Comm}(A, n, m)$.

(3.) *There exists a sub-module $M \subset \mathbb{Z}^n$ and an isomorphic map $\varphi : C_B \to \mathbb{Z}^n/M$ where, $(\mathbb{Z}^n/M, \bigoplus)$ is viewed as a (factor) module with binary addition operation $\bigoplus$, such that for any stream σ,*

$$\varphi(a \oplus_B \sigma) = \varphi(a) \bigoplus \ [f(\sigma)]$$

where, $x \mapsto [x]$ is the canonical homomorphism from $\mathbb{Z}^n$ to $\mathbb{Z}^n/M$ (that is, $[x]$ is the unique coset of M to which x belongs).

(4.) $\textsc{Comm}(B, n, m) = \Theta((n - \dim M) \log m)$, *where,* $\dim M$ *is the dimension of M.*

(5.) $\textsc{Time}(B, n, m) = O(n \times \textsc{Comm}(B, n, m))$.

Conversely, given any sub-module $M \subset \mathbb{Z}^n$*, a stream automaton* $A = (n, C_A, o_a, \oplus_A, \psi_A)$ *can be constructed such that there is an isomorphic map* $\varphi : C_A \to \mathbb{Z}^n/M$ *such that for any stream* σ*,*

$$\varphi(a \oplus_A \sigma) = \varphi(a) \bigoplus [f(\sigma)] \ .$$

where, $\bigoplus$ *is the addition operation of* $\mathbb{Z}^n/M$*,*

$$\begin{aligned} \textsc{Comm}(A, n, m) &= \log \left\lceil \left| \{[x] : x \in \mathbb{Z}^n_{2m+1}\} \right| \right\rceil \\ &= \Theta((n - \dim M) \log m) \end{aligned}$$

and $\textsc{Time}(A, n, m) = O(n \times \textsc{Comm}(A, n, m))$.

4 FDP Algorithm from Data Streaming

In this section we will show that a data streaming algorithm for approximating a function of the frequency vector of a data stream implies the existence of a special kind of FDP[VECSUM] algorithm, called reducible algorithms. We first define this notion.

Notation: Given a vector sequence $\tau = x_1, x_2, \ldots, x_k$ where the x_j's belong to $\mathbb{Z}^n$, we use the following abbreviations. For any function $\phi : \mathbb{Z}^n \to D$, we extend the notation to define $\phi : (\mathbb{Z}^n)^k \to D^k$, for each $k \geq 1$, as follows.

$$\phi(\tau) \stackrel{\text{denote}}{=} (\phi(x_1), \ldots, \phi(x_k)) \ .$$

For $\tau \in (\mathbb{Z}^n)^*$, let

$$\sum \tau \stackrel{\text{denote}}{=} x_1 + x_2 + \ldots + x_k \ .$$

Definition 1. *An* FDP *algorithm* $Q = (\phi, \odot, \psi)$ *is said to be reducible if the function* $\Upsilon : (\mathbb{Z}^n)^* \to M$ *defined as*

$$\Upsilon(\tau) = \odot(\phi(\tau))$$

is symmetric and

$$\begin{aligned} \Upsilon(\tau) &= \Upsilon\big(\textstyle\sum \tau\big), \ \textit{and} \\ \Upsilon(\tau, \sigma) &= \Upsilon(\tau', \sigma), \\ &\quad \forall \sigma, \tau, \tau', \in (\mathbb{Z}^n)^* \textit{ such that } \Upsilon(\tau) = \Upsilon(\tau'). \end{aligned}$$

Reducible algorithms are denoted by the pair (Υ, ψ).

Theorem 3. *Let $g : \mathbb{Z}^n \to O$ and suppose there is a* $\textsc{Comm}(n, m)$*-communication data streaming algorithm for approximating g over the frequency vector $f(\sigma)$ of its input stream σ with respect to an approximation predicate* Approx. *Then there exists a reducible* fdp[vecsum] *algorithm for approximating g with respect to* Approx *that has communication at most* $\textsc{Comm}(n, m)$ *and time $O(n \times \textsc{Comm}(n, m))$.*

Proof. Let $A = (n, C_A, o_A, \oplus_A, \psi_A)$ be a total stream algorithm that approximates $g(f(\sigma))$. Then, by Theorem 2, there exists a stream automaton $B = (n, C_B, o_B, \oplus_B, \psi_B)$ computing the same approximation to $g(f(\sigma))$ (Theorem 2, part (1)), such that $\textsc{Comm}(n, m) \geq \textsc{Comm}(B, n, m)$ (by part (2) of Theorem 2), and there exists a sub-module M of $\mathbb{Z}^n$ and an isomorphism φ from the set of configurations of B to the factor module $(\mathbb{Z}^n/M, \bigoplus)$ that preserves the transition function (by part (3) of Theorem), that is,

$$\varphi(a \oplus_B \sigma) = \varphi(a) \bigoplus [f(\sigma)]$$

where, $[f(\sigma)]$ denotes the coset $f(\sigma) \mod M$, that is, $\{f(\sigma) + y \mid y \in M\}$. The fdp algorithm template $(\phi, \odot, \psi)$ is defined as follows.

$$\begin{aligned} \phi(f) &= [f] \\ \odot([f_1], \ldots, [f_r]) &= [f_1 + \ldots + f_r] \\ \psi([f]) &= \psi_B(\varphi^{-1}([f])) \ . \end{aligned}$$

An Υ function can be defined as $(x_1, \ldots, x_t) \mapsto [x_1 + \ldots + x_k]$ implying that the algorithm is reducible. For any binary tree T, the correctness of $P = (T, \phi, \odot, \psi)$ follows from the commutative, associative addition operation $\bigoplus$ in a module. The communication requirement of the fdp algorithm is at most the communication $\textsc{Comm}(B, n, m)$ which is at most $\textsc{Comm}(A, n, m)$ bits, by Theorem 2, part (2).

The time requirement can be calculated as follows. It is defined as the maximum of the time requirements of the functions ϕ and $\odot$. The ϕ function can be computed by viewing it as a linear mapping $T : \mathbb{Z}^n \to \mathbb{Z}^n/M$, where, M is fixed such that

$$\phi(x) = Tx = x \mod M = [x] \ .$$

By Theorem 2, T can be represented as a matrix with dimensions $(n - \dim M) \times n$. For any x with $\|x\|_\infty \leq m$, computing Tx requires time $O((n -$

$\dim M) \times n \log m) = O(n \times \textsc{Comm}(A, n, m))$. The $\odot$ function is simply the addition of vectors modulo M, and can be performed in essentially linear time $O(n - \dim M)$ simple operations ($+$, $\bmod$) over integers of absolute value at most m.

■

Remark 1. The contribution beyond MUD is that the data streaming algorithm is allowed to compute an approximation to $g(f(\sigma))$ for input σ. For different re-orderings of σ, the approximation computed by the streaming algorithm may be different, that is, the data stream algorithm may not compute a symmetric function of its input stream.

Remark 2. Theorem 3 gives a time bound of $O(n \times \textsc{Comm}(n, m))$, whereas MUD yields a a time bound of $\left(n \cdot 2^{O(\textsc{space}(n,m))^2}\right)$. Since $\textsc{space}(n, m) \geq \textsc{Comm}(n, m)$, this is a significant improvement. The time bound of $O(n \times \textsc{Comm}(n, m))$ is obtained as the worst case bound for computing the reduction function ϕ. The bound for computing the message composition function $\odot$ is $O(\textsc{Comm}(n, m))$, which is smaller. On the contrary, the worst case time requirement for the MUD simulation is obtained for the composition function $\odot$ (the worst case time requirement for the ϕ function is not considered in MUD).

Remark 3. Theorem 3 does not give an explicit bound regarding the space usage of the FDP algorithm. The upper bound on the space usage of the $\odot$ operator is $O(n - \dim M)$, since we only need to add the corresponding coordinates $\bmod M$. To compute ϕ, it suffices to keep the $(n - \dim M) \times n$ matrix T as explained in the proof of Theorem 3. Therefore, the space requirement of the ϕ function is bounded by $O(n(n - \dim M)) = O(n \cdot \textsc{Comm}(n, m))$.

Remark 4. Theorem 3 does not claim a simulation, that is, the translation from the data streaming algorithm to the FDP[VECSUM] algorithm may not be computable. Only existence is asserted.

Remark 5. The FDP[VECSUM] algorithm referred to in Theorem 3 is indeed a MUD algorithm. The theorem therefore shows the existence of a MUD algorithm corresponding to a data streaming algorithm whose output is not necessarily a symmetric function of its input stream.

The converse result, namely, the simulation of data streaming algorithms from FDP algorithms is straightforward, since a stream of m records may be

viewed as a linear tree. The resource bounds of the obtained streaming algorithm are the same as those of the FDP algorithm as noted by [2].

References

[1] Jeffrey Dean and Sanjay Ghemawat. "Simplified data processing on large clusters". In *Int'l Conference on Operating System Design and Implementation OSDI*, 2004.

[2] Jan Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. "On distributing symmetric streaming computations". In *Proceedings of ACM Symposium on Discrete Algorithms (SODA)*, pages 710–719, 2008. Full version at arXiv:cs/0611108v2.

[3] S. Ganguly. "Lower bounds for frequency estimation over data streams". In *Proceedings of the Computer Science Symposium of Russia (CSR), Springer LNCS 5010*, pages 204–215, 2008.