New Algorithms for Distributed Sliding Windows

Sutanu Gayen[1]
University of Nebraska-Lincoln
Lincoln NE, USA
sutanugayen@gmail.com

N. V. Vinodchandran[2]
University of Nebraska-Lincoln
Lincoln NE, USA
vinod@cse.unl.edu

Abstract

Computing functions over a distributed stream of data is a significant problem with practical applications. The *distributed streaming model* is a natural computational model to deal with such scenarios. The goal in this model is to maintain an approximate value of a function of interest over a data stream distributed across several computational nodes. These computational nodes have a two-way communication channel with a *coordinator node* that maintains an approximation of the function over the entire data stream seen so far. The resources of interest, which need to be minimized, are communication (primary), space, and update time. A practical variant of this model is that of *distributed sliding window (dsw)*, where the computation is limited to the last W items, where W is the window size. Important problems such as sampling and counting have been investigated in this model. However, certain problems including computing frequency moments and metric clustering, that are well studied in other streaming models, have not been considered in the distributed sliding window model.

We give the first algorithms for computing the frequency moments and metric clustering problems in the distributed sliding window model. Our algorithms for these problems are a result of a general transfer theorem we establish that transforms any algorithm in the distributed infinite window model to an algorithm in the distributed sliding window model, for a large class of functions. In particular, we show an efficient adaptation of the smooth histogram technique of Braverman and Ostrovsky, to the distributed streaming model. Our construction allows trade-offs between communication and space. If we optimize for communication, we get algorithms that are as communication efficient as their infinite window counter parts (upto polylogarithmic factors).

2012 ACM Subject Classification Theory of computation → Streaming models, Theory of computation → Sketching and sampling, Theory of computation → Distributed algorithms

Keywords and phrases distributed streaming, distributed functional monitoring, distributed sliding window, frequency moments, k-median clustering, k-center clustering

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.22

1 Introduction

Modern data often arrive fast and are distributed over several nodes. In such situations it is not practical to store and process all the data at a central location. The *distributed streaming* model is a natural architecture for such computing scenario. In this model, a set

[1] Research Funded in part by NSF grant CCF-1422668 and UNL Layman Grant
[2] Research Funded in part by NSF grant CCF-1422668 and UNL Layman Grant

of distributed computational nodes get a stream of data items. Each distributed node has a two-way communication channel to a *coordinator node.* The goal of the coordinator node is to continuously keep track of an approximate value of a function of interest over the union of all streams distributed over the nodes. As in the case of traditional streaming situation, it is assumed that a data item cannot be revisited unless stored on local memory. The primary resource of interest is the total number of bits of *communication* between the coordinator node and distributed nodes. Other important resources are total space used over all nodes and the update time per data item.

The focus of this paper is the *sliding window* variation of this distributed streaming model: *distributed sliding window (dsw)* model. In this model, the goal of the coordinator node is to continuously compute an approximation of a desired function over *only the last W data items* in the union of all data items arriving at each of the distributed nodes. Example of practical situations where such computing scenario arise include geographically distributed e-commerce servers or LAN devices or sensor nodes trying to compute some statistics about the transactions made or the traffic passed or the interesting events happened respectively, for the last million items or during last 24 hours. In general the sliding window variations are more difficult for algorithm design than their infinite window counter parts because of 'implicit deletion': the last element of the current window gets deleted at the arrival of the next element. In the distributed model there are additional challenges since the coordinator node is not directly aware of the arrival of a new item.

1.1 Our Results

We design efficient (communication, space, time) algorithms in the distributed sliding window model for functions including frequency moments and metric clustering. Our algorithms for these problems are results of a general transfer theorem we establish that transforms any algorithm in the distributed (infinite window) streaming model (diw) to an algorithm in the distributed sliding window model, for a large class of functions. Specifically, we adapt the *smooth histogram* technique of Braverman and Ostrovsky [5] that applies to the class of *smooth functions* in the single stream model, to the distributed setting. In particular, we prove the following transfer theorem (please refer to the next section for definitions and notations).

▶ **Theorem.** *Let f be an (α, β)-smooth function f for some $0 < \beta < \alpha < 1$. Let $0 < \epsilon < 1$ be such that $b = \frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta) < 1$. Fix any $0 \le x, y \le 1, x + y = 1$. Suppose there is a diw algorithm $\mathcal{B}$ computes f over stream size at most m, upto approximation ratio $(1 \pm \epsilon)$, using cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then there is another algorithm that computes f over a dsw of size W upto approximation ratio $(1 \pm (\alpha+\epsilon))$ using cost $\langle L \cdot W^x \cdot (1 + \log W) c_{\mathcal{B}}(W,\epsilon), (4L \cdot s_{\mathcal{B}}(W,\epsilon) + W^y), L(4 + \log W) t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L = ((\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2)$, $f_{\max} =$ the maximum value of f over any window of size W, and $f_{\min} =$ smallest non-zero value of f. We assume storing each data element takes unit space.*

The construction allows trade-offs between communication and space. In particular setting $x = y = 1/2$, we get that for any function f with $f_{\max}/f_{\min} = \text{poly}(W)$, a diw algorithm for f with cost $\langle c, s, t\rangle$ can be transformed to get a dsw algorithm with cost $\langle \tilde{O}(\sqrt{W}c), \tilde{O}(\sqrt{W} + s), \tilde{O}(t)\rangle$, where $\tilde{O}$ hides polylog factors.

We apply this general algorithm for computing frequency moments and metric clustering problems to get new algorithms that are as communication efficient as their infinite window counter parts (upto polylog factors). Although the cost functions associated with clustering problems are not exactly smooth, we still give dsw algorithms for them based on the single stream clustering algorithms given in [4].

1.2 Previous work

Initial research in the distributed streaming model appeared in papers including [2, 10, 15, 18, 19]. These papers designed distributed streaming algorithms for several natural problems including approximately tracking the functions: sum, top-k frequencies, set-expression cardinality, approximate quantiles and thresholded counts. We refer the reader to the above papers for practical motivations behind this model. A formal algorithmic approach towards distributed streaming was first given by Cormode, Muthukrishanan and Yi [11]. They called their model *distributed functional monitoring model* where the task is to continuously monitor whether the function value is $\geq \tau$ or $\leq (1-\epsilon)\tau$ for a threshold τ and an error parameter[3] ϵ. They designed the first algorithm for F_p, the p^{th} frequency moment and also proved lower bound results on the communication cost of monitoring F_p for $p \leq 2$. Efficient randomized algorithms for monitoring count, frequencies and ranks were given in [17]. Woodruff and Zhang [22] designed better algorithms for F_p and provided matching lower bound for the communication cost. Sampling algorithms over distributed streams were given in [12, 21]. Recently Chen and Zhang [7] gave an algorithm for distributed monitoring of entropy. We refer the reader to [9] for a survey by Cormode on this topic.

While the "infinite-window" distributed streaming has received considerable attention, its sliding (or finite) window counterpart has received only limited attention. The first paper to deal with distributed stream processing over a finite sliding window is [12] where the authors present algorithms for sampling that is efficient in communication, space and time. Later, in [6, 14], efficient algorithms for distributed sliding window were designed for the problems of counting the number of bits, quantiles, and heavy hitters. In [20], the authors extend the count-min sketch algorithm to the dsw model. In [16], the present authors give an algorithm for *Euclidean* k-median clustering problem. To the best of our knowledge computations of F_p for a general p and metric clustering are not yet considered in this model (which we consider in this paper). We would like to note that F_p computation and clustering problems have received considerable attention in the *single stream sliding window models* [1, 3, 4, 5, 8].

2 Background and definitions

2.1 The models

The distributed streaming model. In the distributed streaming model there are $(K+1)$ computational nodes: $\{N_1, N_2, \ldots, N_K, C\}$ where N_is are called *distributed* nodes and C is called the *coordinator* node. These nodes have to collectively compute a function f over a global stream of data items: $\{d_1, d_2, \ldots, d_t, \ldots, d_N\}$ which are distributed over N_is in an arbitrary manner. More precisely, at time t, the item d_t will be sent to the node N_j for some $1 \leq j \leq m$. At all times t, the coordinator should maintain an approximation of f over the set of items $\{d_1, d_2, \ldots, d_t\}$ seen so far from the global stream. In order to achieve this, each N_j can communicate with C through a bi-directional channel. An algorithm in this model must work for any ordering of the global input stream. The resources of interest are the total communication, total space usage over all nodes, and time to process each data item. The term *local stream* will be used to refer to the sub-stream seen only at a particular node N_j. In this paper we call this model is the *distributed infinite window* model.

[3] This does not loose generality as the authors observed that any monotonic function f can be computed continuously using $\tilde{O}(\frac{1}{\epsilon})$ copies of a monitoring algorithm for f.

The distributed sliding window model. In the sliding window variation of the distributed infinite window model, there are the global stream and the set of $(K+1)$ nodes as before. But at any time t, the coordinator needs to maintain an approximation of the function f over the set of *most recent* W data items: $\{d_{t-W+1}, \dots, d_t\}$. This set of items is known as the *active window* and W is known as the *window size*. As in most of the prior literature, we assume each data item comes with a unique (modulo W) time-stamp[4].

2.2 Smooth functions and smooth histograms

The notion of smooth functions was introduced by Braverman and Ostrovsky in [5]. The main property that a smooth function f should satisfy is the following continuity property: Consider f computed on a stream starting at two time points (or indices) i and j ($j > i$). If f computed starting at i and f computed starting at j are within a constant factor of each other at a given point in time, then it will remain within a constant factor in the future.

▶ **Definition 1** ((α, β)-smooth function [5])**.** A function f defined on a set χ of elements is called (α, β)-smooth, for some $0 < \beta < \alpha < 1$ if (1) f is non-decreasing and non-negative, (2) $f(A)$ is at most $poly(|A|)$, (3) $(1-\beta)f(A \cup B) \leq f(B) \implies (1-\alpha)f(A \cup B \cup C) \leq f(B \cup C)$ for any set $A, B, C \subseteq \chi$.

Braverman and Ostrovsky show that for such smooth functions, it suffices to run an online algorithm to compute the function starting at logarithmic number of carefully chosen indices to get constant approximation at any given window. These indices correspond to a constant factor decrease in the value of the function. The corresponding data structure is referred as smooth histogram. This resulted in a construction that translates any single stream infinite window algorithm to a single stream sliding window algorithm for smooth functions with comparable space complexity (up to log factors) as that of the infinite window algorithm.

▶ **Definition 2** (Approximate smooth histogram [5])**.** Let f be an (α, β) smooth function. A smooth histogram for f is a data structure that consists of an increasing set of indices $[I_1, I_2, \dots, I_L]$ over a sliding window of size W with the following properties.

1. For each $i = 1$ to L, there is an instance of a $(1 \pm \epsilon)$-approximating algorithm A, running for approximating $f(I_i, N)$, the value of f for $\epsilon \leq \beta/4$ over the set $\{d_{I_i}, d_{I_i+1}, \dots, d_N\}$, where d_j is the data element at location j and d_N is the most recently arrived element.
2. I_1 is expired and I_2 is active or $I_1 = 0$.
3. For $i = 1$ to $t-2$, either 1) $(1-\alpha)f(I_i, N) \leq f(I_{i+1}, N)$ and $(1-\beta/2)f(I_i, N) > f(I_{i+2}, N)$ or 2) $(1-\beta/2)f(I_i, N) > f(I_{i+1}, N)$ and $I_{i+1} = I_i + 1$.

For a smooth function, the third point above ensures that consecutive indices are farthest apart but staying within at least $(1-\alpha)$ factor (or immediate in stream and drops by $> (1-\beta/2)$ factor). Note that $f(I_i, N)$ where I_i is the least index from the histogram that is contained in the current window, approximates the value of the function on the current window.

2.3 Notation

In this paper we use the abbreviations diw and dsw to mean distributed infinite window and distributed sliding window respectively. We denote by $(t_1, t_2]$ the subset of the stream from

[4] This model is also known as time-based dsw model. There is another variation called sequence-based dsw, where no time-stamps are available. This model is harder to design algorithms on.

$(t_1 + 1)$-st through t_2-th element. We also use: K to denote the number of distributed nodes, m the length of stream, W the length of window and k the number of medians/centers for clustering. $\langle c, s, t\rangle$ denotes the costs of our diw/dsw algorithm, where c is the communication complexity over any window of size W or over the length of the stream m as appropriate, s is the space complexity and t is the update time (possibly amortized). For a function f, by a c-factor approximation we mean $\frac{f}{c} \le \tilde{f} \le c \cdot f$ and by $(1 \pm \epsilon)$ approximation we mean $(1-\epsilon)f \le \tilde{f} \le (1+\epsilon)f$.

3 A transfer theorem for smooth functions

In this section we give a general construction that transforms a distributed infinite window (diw) algorithm for smooth function to a distributed sliding window (dsw) algorithm. We assume that the infinite window algorithm $\mathcal{B}$ has a cost-tuple $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$, over a stream of length m and with error parameter ϵ. In particular we prove the following theorem (same as the theorem stated in the introduction).

▶ **Theorem 3.** *Let f be an (α,β)-smooth function f for some $0 < \beta < \alpha < 1$. Let $0 < \epsilon < 1$ be such that $b = \frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta) < 1$. Fix any $0 \le x, y \le 1, x + y = 1$. Suppose there is a diw algorithm $\mathcal{B}$ computes f over stream size at most m, upto approximation ratio $(1 \pm \epsilon)$, using cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then there is another algorithm that computes f over a dsw of size W upto approximation ratio $(1\pm(\alpha+\epsilon))$ using cost $\langle L \cdot W^x \cdot (1+\log W)c_{\mathcal{B}}(W,\epsilon), (4L \cdot s_{\mathcal{B}}(W,\epsilon) + W^y), L(4+\log W)t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L = ((\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2)$, $f_{\max}$ = the maximum value of f over any window of size W, and $f_{\min}$ = smallest non-zero value of f. We assume storing each data element takes unit space.*

Note that the construction allows trade-offs between communication and space. In particular setting $x = y = 1/2$, we get that for any function f with $f_{\max}/f_{\min} = \mathrm{poly}(W)$, a diw algorithm for f with cost $\langle c, s, t\rangle$ can be transformed to get a dsw algorithm with cost $\langle \tilde{O}(\sqrt{W}c), \tilde{O}(\sqrt{W} + s), \tilde{O}(t)\rangle$, where $\tilde{O}$ hides polylog factors.

We first give the algorithm and its proof for the case when $x = 0$ and $y = 1$, and then point out how to modify this algorithm to get the general algorithm. This algorithm has communication and time costs almost same as (upto polylog(W)) that of the diw algorithm, but uses $\Theta(W)$ space.

▶ **Theorem 4.** *Let f be an (α,β) smooth function for some $0 < \beta < \alpha < 1$. Let $0 < \epsilon < 1$ be any number such that $b = \frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta) < 1$. Further assume f has a $(1\pm\epsilon)$-approximate diw algorithm $\mathcal{B}$ over stream size at most m, with cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then, there is a dsw algorithm for computing f upto approximation ratio $(1\pm(\alpha+\epsilon))$ with cost $\langle L(\log W + 1)c_{\mathcal{B}}(W,\epsilon), L \cdot s_{\mathcal{B}}(W,\epsilon) + W, L(\log W + 1)t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L \le (\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2$, $f_{\max}$ = the maximum value of f over any window of size W, and $f_{\min}$ = smallest non-zero value of f. We assume storing each data element takes unit space.*

High level idea of the algorithm: Our general approach is to adapt the smooth histogram technique for sliding windows due to Braverman and Ostrovsky [5] to the distributed setting. Braverman and Ostrovsky showed that for smooth functions, if streaming algorithms are maintained from a small set of carefully chosen indices of the active window, one of these algorithms would approximate the value of the function over all active windows in near future. These indices correspond to a drop of the value of the function by some constant factor. Their algorithm has three main steps: when a new data item d arrives (1) start a new instance of the streaming algorithm $\mathcal{A}$ from this new item (2) update all running instances of

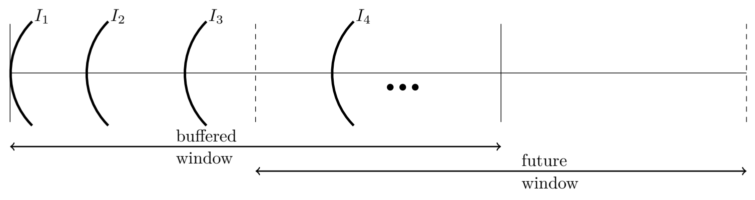

Figure 1 Online algorithms run from each index I_j (position indicated by '('). The one from I_3 can be used to approximate f over the future window. Thus, indices from the buffered window serves for next W elements.

$\mathcal{A}$ with d and (3) remove redundant indices and the corresponding instances of $\mathcal{A}$. There are technical challenges to translate the smooth histogram technique to the distributed setting. The main obstacle is the following: In the single stream case, for each newly arrived element, an instance of $\mathcal{A}$ is started from its index. Most of them are removed at some later point in the future so that at all times only a logarithmic number of indices are kept. In distributed setting, if one has to start instances of $\mathcal{A}$ each time a new element arrives, it will cost $\Omega(W)$ bits of communication (from the distributed node where the item arrives to the coordinator).

In order to reduce the communication cost we use the following approach. Instead of continuously building the histogram, we observe that it is enough to create it once per W items. Once built, this histogram will continue to work till the arrival of next W^{th} item due to smoothness of the function. In other words, steps (1) and (3) in the previous discussion could be dropped except once per W items. This keeps the asymptotic time and communication cost small. We split the entire stream into static windows of size W: $(0, W], (W, 2W], \ldots, (aW, aW + W], \ldots$ and we build the smooth histogram only when the current window coincides with one of the static windows. This is done by buffering the entire expiring window using $O(W)$ space (see Figure 1 for illustration). The indices, which correspond to a drop of some constant factor for f, are obtained by performing binary searches on the buffered static window. This will introduce further communication cost and time for about $O(\log^2 W)$ many instances of the online algorithm.

Proof. (*Of Theorem 4*). A high-level pseudocode of the algorithms is given in Algorithm 1. We split the entire stream into static windows of size W: $(0, W], (W, 2W], \ldots, (aW, aW+W], \ldots$ and maintain a smooth histogram over exactly one of them. If the current time t satisfies $aW - W < t < aW$, the smooth histogram from the static window $(aW - 2W, aW - W]$ can be used to approximate $f([t - W + 1, t])$. We recall, the smooth histogram maintains online algorithms from a set of appropriately chosen indices guarantees that any two consecutive indices are either consecutive or are $(1-\alpha)$-factor close in f. If $I_1 \leq (t - W + 1) < I_2$ are the two unique consecutive pair of indices enclosing $(t - W + 1)$, the value of the online algorithm from the index I_1 can be used to approximate $f([t - W + 1, t])$ upto $(1 \pm (\alpha + \epsilon))$-factor. We buffer the next static window $(aW, aW + W]$ locally (at the distributed nodes as they arrive) and build the smooth histogram from it at time $(aW + W)$. We describe how to build this in detail in the following claim.

▶ **Claim 5.** *Suppose we have stored a static window $[\lambda, \rho] = (aW, aW + W]$ locally. A smooth histogram $\mathcal{H}$ for $[\lambda, \rho]$ can be built using cost $\langle L \log W \cdot c_{\mathcal{B}}(W, \epsilon), (s_{\mathcal{B}}(W, \epsilon) + W), L \log W \cdot t_{\mathcal{B}}(W, \epsilon)\rangle$, where the number of indices in $\mathcal{H}$ is $L \leq (\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2$.*

Algorithm 1: High level dsw algorithm for smooth functions

```
   Input: A stream of data: $\langle d_1, d_2, \ldots, d_N \rangle$
   Output: Approximate value of $f((N-W, N])$
1  while Not the end of stream do
2      Let $d_N$ be the newly arrived item;
3      if $N = aW + W$ then
4          Delete the current smooth histogram $\mathcal{H}((aW - W, aW])$;
5          $\mathcal{H}((aW, aW + W]) \leftarrow$ Build a new smooth histogram for $(aW, aW + W]$;
6          Output the value of online algorithm over $(aW, aW + W]$;
7      else
8          if $aW < N < aW + W$ then
9              Update the online algorithm from each index of $\mathcal{H}((aW - W, aW])$ with
                $d_N$;
10             Buffer $d_N$ at the node where it arrived;
11             $I_j, I_{j+1}$ be the immediate indices such that $I_j \leq N - W + 1 < I_{j+1}$;
12             Output the value of the online algorithm started from index $I_j$;
13         end
14     end
15 end
```

Proof. We find the indices from the buffered window $[\lambda, \rho]$ by binary search and by running the online algorithm $\mathcal{B}$ on the buffered set of items[5]. The first index is always λ. The last index is always the index ρ. We denote by $\tilde{f}$, a $(1 \pm \epsilon)$ factor approximation for f, found using $\mathcal{B}$. Then, the second index is created at a time-stamp t, such that,

$$\tilde{f}([t, \rho]) \geq (1 - \beta)\frac{(1+\epsilon)}{(1-\epsilon)}\tilde{f}([\lambda, \rho]). \tag{1}$$

This ensures, $f([t, \rho]) \geq (1 - \beta)f([\lambda, \rho])$, as desired for smooth histogram. In fact, we try to find such a t as far as possible in the window to minimize the number of indices. We tag a time-stamp 'yes' if it satisfies Equation (1) and 'no' otherwise. We set two variables $l = \lambda$ and $r = \rho$ and maintain the invariant that l has 'yes' tag and r has 'no' tag. We next check whether $mid = (l + r)/2$ has 'yes' tag or 'no' tag and update l or r appropriately to maintain the invariant. Then, in $\log W$ steps, we will be able to get hold of a t, such that, t has 'yes' tag but $(t + 1)$ has 'no' tag. This is our next index. We find subsequent indices in similar manner.

Notice that, at the $(t + 1)$-st item, the value of $\tilde{f}$ drops at least by factor $(1 - \beta)\frac{(1+\epsilon)}{(1-\epsilon)}$ (Recall, if the indices are consecutive, drop could be larger). This implies, f drops at least by factor $b = (1 - \beta)\frac{(1+\epsilon)^2}{(1-\epsilon)^2} < 1$. Suppose there are L indices in total. Then, After crossing the $(L - 1)$-st index, the value of $\tilde{f}$ is at most $f_{\max} b^{(L-2)}$, where $f_{\max}$ is the maximum

[5] There are some details for running $\mathcal{B}$ on the buffered items. For a general function, assume, all the nodes have some global knowledge of time. Then, a fixed time interval of sufficient length can be allotted for processing each data item. Thus, for example, it may be agreed upon that the k^{th} data item d in the current window will be processed during time interval $(t, t + \Delta \cdot k)$ where Δ is at least as large as the update time of the algorithm and t is the time of arrival of the oldest element of the current window. During this interval, whichever node has received d, will process it. For a permutation-invariant function such as F_p and clustering, the nodes can take turns and run the online algorithm over the desired sub-window of the local stream, to compute the function over any sub-window of the global stream.

value of f over any window. Moreover, assuming the least non-zero value of f is $f_{\min}$, $f_{\max}b^{(L-2)} \geq f_{\min}(1-\epsilon)$. Hence $L \leq (\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2$. We denote by I this set of indices. This concludes the updating of the smooth histogram. While finding the indices, at most $L \log W$ instance of $\mathcal{B}$ are run for at most W units of time. This can be achieved using $L \log W \cdot c_{\mathcal{B}}(W,\epsilon)$ total communication and $LW \log W \cdot t_{\mathcal{B}}(W,\epsilon)$ total time (i.e. amortized update time $L \log W \cdot t_{\mathcal{B}}(W,\epsilon)$ per item). During the binary search, we need space for running at most a single instance of $\mathcal{B}$ at any point in time, and the space is reused. We also need space for buffering the current window. So, the space complexity for this part is $(s_{\mathcal{B}}(W,\epsilon) + W)$. ◀

Afterwards, we maintain $\mathcal{B}$ from each of the indices and update them with newly arrived items. For any time till the arrival of next $(W-1)$ items, let $f_1 > f_2$ be the value of f at the two enclosing indices of the current window. Let $f_{current}$ be the value of f over the current window. From the properties of smooth histogram, either $f_{current} = f_1$, or $(1-\alpha)f_1 < f_2 \leq f_{current} \leq f_1$. Moreover, the value of online algorithm at the former index holds a value $\tilde{f}_1$, such that, $(1-\epsilon)f_1 \leq \tilde{f}_1 \leq (1+\epsilon)f_1$. Hence, $(1-\epsilon)f_{current} \leq \tilde{f}_1 \leq \frac{(1+\epsilon)}{(1-\alpha)} f_{current}$. This is close to $(1 \pm (\alpha+\epsilon))$-approximation for small α and ϵ. We also need to continue running $\mathcal{B}$ from each index for W units of time. This costs at most $L \cdot c_{\mathcal{B}}(W,\epsilon)$ communication, $L \cdot t_{\mathcal{B}}(W,\epsilon)$ update time per item and $L \cdot s_{\mathcal{B}}(W,\epsilon)$ space in total. ◀

Proof of Theorem 3. The proof follows closely from that of Theorem 4. In this case, we break the current window of size W into W^x blocks, each of size W^y, such that $W = W^x \cdot W^y$, for some $0 < x, y < 1, x+y = 1$. We rebuild the smooth histogram per arrival of W^x elements in the combined stream. Then, the nodes need to store at most W^x items. As before, total number of indices within each block, $L_1 \leq L = ((\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2)$. As in the algorithm of Theorem 4, finding these indices is done by binary search, using at most $L_1 \log W$ calls to $\mathcal{B}$. In total, this can be done with $L_1 \log W c_{\mathcal{B}}(W,\epsilon)$ total communication per block (i.e $L_1 \cdot W^x \cdot \log W c_{\mathcal{B}}(W,\epsilon)$ in total), $(s_{\mathcal{B}}(W,\epsilon) + W^y)$ space (since space is reused during binary-search) and $L_1 \log W W^y t_{\mathcal{B}}(W,\epsilon)$ total time per block (i.e. amortized $L_1 \log W t_{\mathcal{B}}(W,\epsilon)$ time per item).

Subsequently, we need to maintain $L_1 \cdot W^x$ online algorithms from each of the indices within the current window of size W. But we can do better by removing unnecessary ones while introducing indices from a new block. We arrange the combined indices in decreasing order of arrival. Then, for each index i, we look for the subsequent index j where the current value of the online algorithm drops by factor b for the first time. We remove all indices strictly between i and $(j-1)$ if there are any. We repeat this removal procedure until no more indices can be removed in such a manner. Then, starting from each index, at the next to next index, value of online algorithm drops at least by factor b. After merging, there are $L_2 \leq (2(\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2) \leq 2L$ such indices at any point in time. Moreover, by previous discussion, every next index is either the subsequent item, or within a factor b from the previous index. This entire removal takes time linear in the set of the indices to be merged, i.e. at most $10(\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b})$ time per block. So, we ignore this while computing the update time per item. Note that, during this removal, no further communication or space is required. This concludes the indices removal procedure. Then, online algorithms from each of the indices run for $\geq W^y$ and $\leq W$ time. So the total communication cost is at most $L_1 \cdot W^x c_{\mathcal{B}}(W,\epsilon)$. The space complexity for running the online algorithms is at most $2L_2 \cdot s_{\mathcal{B}}(W,\epsilon)$. Using the indices removal procedure, we improve the update time to $2L_2 \cdot t_{\mathcal{B}}(W,\epsilon)$ per arrival, for updating each of the current instances of $\mathcal{B}$. ◀

3.1 Better and simpler algorithm for symmetric smooth functions

For symmetric smooth functions we get simpler algorithm with slightly better cost. In particular, the cost of the algorithm will be $\langle (L+1)\cdot c_{\mathcal{B}}(W,\epsilon)\cdot W^x, 4L\cdot s_{\mathcal{B}}(W,\epsilon)+W^y, 4(L+1)t_{\mathcal{B}}(W,\epsilon)\rangle$.

We call a function symmetric if its value is invariant to the permutation of its arguments. For symmetric functions, we create the indices of the smooth histogram by making a single backward pass (i.e. from the most recent to the least recent item in the window) of the distributed online algorithm on the buffered window $[aW+1, aW+W]$. Let $\tilde{f}(A)$ be the value of this algorithm, which is within $(1\pm\epsilon)$-factor of $f(A)$. We create the last index of the smooth histogram at $(aW+W)$. Recursively assume, the previous index we created was at t_1. During the backward pass, suppose at the time-stamp $(t_2-1)\leq t_1$, the value $\tilde{f}([t_2-1, aW+W])$ is at least $\frac{1}{b}\cdot\tilde{f}([t_1,aW+W]) = \frac{1}{(1-\beta)}\frac{(1-\epsilon)}{(1+\epsilon)}\cdot\tilde{f}([t_1,aW+W])$ for the first time. If this happens at $t_2-1=t_1-1$, we create an index at (t_1-1). Otherwise, we create at t_2, which satisfies $\tilde{f}([t_2,aW+W]) < \frac{1}{b}\tilde{f}([t_1,aW+W])$. This implies, $f([t_1,aW+W]) \geq (1-\beta)f([t_2,aW+W])$, ensuring the smoothness condition between the consecutive indices t_1 and t_2. We find all the L indices in similar manner, where $L = ((\log\frac{f_{\max}}{f_{\min}(1-\epsilon)}/\log\frac{1}{b})+2)$, $f_{\max}$ = the maximum value of f over any window of size W, and $f_{\min}$ = smallest non-zero value of f. This improves the cost of Theorem 4 to $\langle (L+1)c_{\mathcal{B}}(W,\epsilon), (L+1)\cdot s_{\mathcal{B}}(W,\epsilon)+W, (L+1)t_{\mathcal{B}}(W,\epsilon)\rangle$. We can shave off a $\log W$ factor from the costs of Theorem 3 using a similar simpler algorithm. Note that, we crucially use the symmetric nature of the function in the use of the backward online algorithm. The F_p and clustering costs are symmetric functions, whereas, the function 'length of longest increasing subsequence' is asymmetric smooth [5].

4 Applications: Computing F_p and Clustering

In this section we apply the transfer theorem to get new dsw algorithms for approximating F_p and metric clustering problems.

4.1 Computing F_p

▶ **Definition 6** (p^{th} Frequency moment). Given a set of items $\{1,2,\ldots,n\}$ such that their frequencies are $\{f_1,f_2,\ldots,f_n\}$ respectively, their p^{th} frequency moment is defined as $F_p = \sum_{i=1}^{n} f_i^p$.

We first recall a result that shows F_p is smooth.

▶ **Theorem 7** (Lemma 5 of [5]). *Fix any $0<\epsilon<1$. For $p<1, F_p$ is (ϵ,ϵ)-smooth function. For $p\geq 1, F_p$ is $(\epsilon,\frac{\epsilon^p}{p^p})$-smooth function.*

The first distributed functional monitoring algorithm for F_p was given in [11]. In the monitoring model, in a distributed stream of items, one has to decide whether $F_2\geq\tau$ or $F_2\leq(1-\epsilon)\tau$ at all times, for some ϵ and a threshold τ specified. The same paper ([11]) also observed that any monotonic function f can be computed continuously using $\tilde{O}(\frac{1}{\epsilon})$ copies of a monitoring algorithm for f. Later on Woodruff and Zhang provided the following online algorithm for F_p, for any $p\geq 1$. They also showed this algorithm has optimal dependence on K.

▶ **Theorem 8** (Follows from Theorem 8 of [22]). *For any $0<\epsilon<1$, there is an algorithm that continuously computes F_p for any constant $p\geq 1$, over universe $[n]$ over a distributed stream of length at most W upto approximation $(1\pm\epsilon)$ with high probability using cost $\langle\tilde{O}(\frac{K^{p-1}}{\epsilon^{\Theta(p)}}), \tilde{O}(\frac{nK}{\epsilon}), \tilde{O}(\frac{1}{\epsilon^2})\rangle$.*

From Theorem 3, we get the following algorithm for computing F_p, which is $(\epsilon, \epsilon^p/p^p)$-smooth.

▶ **Corollary 9.** *Fix any $0 \leq x, y \leq 1, x+y=1$. For any constant p, there is an algorithm that continuously computes F_p over a time based dsw of width W upto approximation ratio $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}(W^x \frac{K^{p-1}}{\epsilon^{\Theta(p^2)}}), \tilde{O}(W^y + \frac{nK}{\epsilon^{\Theta(p)}}), \tilde{O}(\frac{1}{\epsilon^{\Theta(p)}})\rangle$.*

In particular for F_2, we chose to work with the following result of [11] since its communication cost has much smaller dependence on ϵ.

▶ **Theorem 10** (Follows from Theorem 6.1 of [11])**.** *For any $0 < \epsilon < 1$, there is an algorithm that continuously computes F_2 over universe n over a distributed stream of length at most W upto approximation $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}((\frac{K^2}{\epsilon^2} + \frac{K^{1.5}}{\epsilon^4})), \tilde{O}(\frac{K}{\epsilon^3}), \tilde{O}(\frac{1}{\epsilon^3})\rangle$.*

From Theorem 3, we get the following algorithm for computing F_2, which is $(\epsilon, \epsilon^2/4)$-smooth.

▶ **Corollary 11.** *Fix any $0 \leq x, y \leq 1, x+y=1$. There is an algorithm that continuously computes f over a time based dsw of width W upto approximation ratio $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}(W^x(\frac{K^2}{\epsilon^4} + \frac{K^{1.5}}{\epsilon^8})), \tilde{O}(W^y + \frac{K}{\epsilon^6}), \tilde{O}(\frac{1}{\epsilon^6})\rangle$*

4.2 Metric clustering

In this section we apply the generic algorithm from Section 3 for the functions: k-median and k-center clustering.

▶ **Definition 12** (Metric k-median clustering problem)**.** Given a set of points P from a metric space χ, output $C^* = \arg\min_{C \subseteq \chi, |C| \leq k} \sum_{p \in P} \min_{c \in C} d(p,c)$ and $\mathrm{OPT}_k = \sum_{p \in P} \min_{c \in C^*} d(p,c)$ where d is the distance function of χ.

▶ **Definition 13** (Metric k-center clustering problem)**.** Given a set of points P from a metric space χ, output $C^* = \arg\min_{C \subseteq \chi, |C| \leq k} \max_{p \in P} \min_{c \in C} d(p,c)$ and $\mathrm{OPT}_k = \max_{p \in P} \min_{c \in C^*} d(p,c)$ where d is the distance function of χ.

In this section, approximation ratio of $r > 1$ will mean the clustering cost of the algorithm is in the range $[\mathrm{OPT}, r.\mathrm{OPT}]$. We assume each point takes $O(1)$ space. These two clustering problems have approximation ratio $\Theta(1)$. So a straightforward combination of the local clusterings results in an overall approximation ratio of $O(m)$. The cost functions OPT_k are neither smooth [4].

4.2.1 Metric k-median clustering

We use the following diw algorithm for k-median clustering.

▶ **Theorem 14** (Theorem 2 of [16] restated)**.** *There is a distributed online algorithm for $O(1)$-approximate metric k-median with success probability $(1 - \frac{1}{\mathrm{poly}(W)})$, and with cost $\langle O(kK \log^3 W), O(kK \log W)), O(k \log W)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

We cannot directly apply Theorem 3 since the k-median clustering cost is not smooth. We use an additional property of k-median cost observed in [4] and use ideas from Theorem 4 to get the following theorem, whose proof is in Section 5.

▶ **Theorem 15.** *There is a dsw algorithm for $O(1)$-approximate metric k-median with success probability $(1 - \frac{1}{\mathrm{poly}(W)})$ per W items, and with cost $\langle O(k^2 K \log^5 W), O(k^2 K \log^3 W + W), O(k^2 \log^3 W)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

4.2.2 Metric k-center clustering

We use the following diw algorithm for k-center clustering.

▶ **Theorem 16** (Theorem 6 of [13] restated). *For any $\epsilon > 0$, there is a deterministic distributed online algorithm for $(2+\epsilon)$-approximate metric k-center with cost $\langle O(\frac{kK}{\epsilon}\log W), O(\frac{kK}{\epsilon}\log \mathrm{OPT}), O(k)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

The k-center clustering cost is not smooth. We prove an additional property of k-median cost and use ideas from Theorem 4 to get the following theorem, whose proof is in Section 5.

▶ **Theorem 17.** *There is a deterministic dsw algorithm for $O(1)$-approximate metric k-center with cost $\langle O(k^2K\log^4 W), O(k^2K\log^2 W + W), O(k^2\log^2 W)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

5 Proofs of clustering results

It was shown in [4], the k-median cost behaves like a smooth function if the following additional property is satisfied. For convenience, we abuse the notation of Definition 1 for a smooth function in this subsection, by replacing $(1-\alpha)$ by $\frac{1}{\alpha}$ and $(1-\beta)$ by $\frac{1}{\beta}$ for appropriate $\alpha > \beta > 1$. We denote by $Cost(P, O)$ the k-median clustering cost for a set of points P, when O is the set of k medians. For convenience, we drop the k in OPT_k when there is no ambiguity.

▶ **Lemma** (Lemma 3.1 of [4] restated). *For any distinct sets of points $A, B, C \subseteq \chi$ from some metric space χ, $\mathrm{OPT}(A\cup B) \leq \gamma\mathrm{OPT}(B) \implies \mathrm{OPT}(A\cup B\cup C) \leq (2+r\gamma)\mathrm{OPT}(B\cup C)$ for any $r, \gamma \geq 1$, provided the following property holds for the sets A, B: There exists a k-median clustering $t : (A\cup B) \to F$ upto approximation ratio r such that $|t^{-1}(f)\cap A| \leq |t^{-1}(f)\cap B|$ for each median $f \in F$.*

▶ **Theorem.** *There is a dsw algorithm for $O(1)$-approximate metric k-median with success probability $(1-\frac{1}{\mathrm{poly}(W)})$ per W items, and with cost $\langle O(k^2m\log^5 W), O(k^2m\log^3 W + W), O(k^2\log^3 W)\rangle$ assuming $\mathrm{OPT} = \mathrm{poly}(W)$.*

Proof. We split the stream into static windows of the form: $[aW+1, aW+W]$ and store this window locally. At time $(aW+W)$, we need to rebuild a smooth histogram. For this, we use the slightly better and simpler algorithm from the remark in Section 3.1. We run $\mathcal{A}$ from Theorem 14 backwards (i.e. from item $(aW+W)$ to item $(aW+1)$). Let the approximation factor of $\mathcal{A}$ be λ. The last index of the smooth histogram is at time-stamp $(aW+W)$. Suppose, the last time we created an index at time-stamp t_l and the value of $\mathcal{A}$ at t_l was $v_l \in [\mathrm{OPT}, \lambda\mathrm{OPT}]$. Fix any $\gamma > \lambda$. Let $(t_{l-1}-1)$ be the time when value of $\mathcal{A}$ is at least $\frac{\gamma}{\lambda}v_l$ for the first time. If $t_{l-1}-1 = t_l-1$, we create the next index at (t_l-1). Otherwise at t_{l-1}, $\mathcal{A}([t_{l-1}, aW+W]) < \frac{\gamma}{\lambda}v_l$. This implies, $\mathrm{OPT}([t_{l-1}, aW+W]) \leq \gamma\cdot\mathrm{OPT}([t_l, aW+W])$. This t_{l-1} is our next index. In total, at most L such indices will be created, where $L = ((\log\frac{\lambda f_{\max}}{f_{\min}})/\log\frac{\gamma}{\lambda})+2) = O(\log W)$, $f_{\max} =$ the maximum value of k-median cost over any window of size W, and $f_{\min} =$ smallest non-zero value of k-median cost. We refer to these set of indices as 'outer' indices. For each outer index, we also record the set of k medians produces by $\mathcal{A}$. Let the indices be $I = \{I_1 < I_2 < \cdots < I_L\}$ and the corresponding sets of k medians be $C = \{C_1, C_2, \ldots, C_L\}$, where each $C_i = \{c_{i1}, c_{i2}, \ldots, c_{ik}\}$. We communicate I and C to each node. We claim that for any $I_i \leq t < I_{i+1}$, C_i is a $\gamma\lambda$-approximate set of k-medians for $[t, aW+W]$. This is because, $Cost([t, aW+W], C_i) \leq Cost([I_i, aW+W], C_i) \leq \lambda\cdot OPT([I_i, aW+W]) \leq \gamma\lambda\cdot OPT([I_{i+1}, aW+W]) \leq \gamma\lambda\cdot OPT([t, aW+W])$ (Using monotonicity of OPT and smoothness).

We also need to ensure the additional property from Lemma 5. We ensure this by keeping a set of 'inner' indices between each pair of outer indices. We describe below how to find the inner indices between I_1 and I_2. Other inner indices can be found accordingly. For any $i = 1$ to L, and for any set $S = [t, aW + W] \subseteq [I_i, aW + W]$, let n_{ij}^S denote the number of points from S which map to the median c_{ij}, in the clustering $[I_i, aW + W] \to C_i$. Let $\tilde{n}_{ij}^S$ denote a $(1 \pm \frac{1}{10})$ approximation of n_{ij}^S. We first assume the coordinator can compute $\tilde{n}_{ij}^S$ for any $S = [t, aW + W] \subseteq [I_i, aW + W]$. We defer the description of how to compute this in the following paragraph. Let J_1 be the first (earliest in window) inner index between I_1 and I_2. We will create the index J_1 at the farthest time-stamp $I_1 < t \leq I_2$, such that

$$\forall j = 1 : k, \tilde{n}_{1j}^{[I_1, aW+W]} \leq (18/11) \cdot \tilde{n}_{1j}^{[t, aW+W]} \tag{2}$$

The later condition ensures, for each median $c_{1j} \in C_1$, $n_{1j}^{[I_1,aW+W]} \leq 2 \cdot n_{1j}^{[t,aW+W]}$, equivalently $n_{1j}^{[I_1,t-1]} \leq n_{1j}^{[t,aW+W]}$, as demanded in the additional property from Lemma 5. Such a farthest t satisfying Equation 2 is found by using binary search. Notice that $t = l := I_1$ is always satisfied. We first guess $t = r := I_2$. If this t satisfies Equation 2, we already have an index at I_2 and we don't need to keep any inner index. If not, we next guess $t = \lceil \frac{l+r}{2} \rceil$ and if this t satisfies, we change $l = \lceil \frac{l+r}{2} \rceil$, otherwise, we change $r = \lceil \frac{l+r}{2} \rceil$. In this way, we preserve the invariant that l satisfies Equation 2 but r does not. In $O(\log W)$ steps, we will get a t^*, such that t^* satisfies but $(t^* + 1)$ does not. We set $J_1 = t^*$. We find the next inner index J_2 similarly using the same clustering C_1 and at the farthest time-stamp t, such that, $\forall j = 1 : k, \tilde{n}_{1j}^{[J_1,aW+W]} \leq \frac{18}{11} \tilde{n}_{1j}^{[t,aW+W]}$ holds. Note that, the set C_1 is a $\gamma\lambda$ approximate median for any $I_1 \leq t < I_2$, from previous discussion. So, the additional property from Lemma 5 holds at indices J_1 and J_2, with respect to the clustering C_1 and $r = \gamma\lambda$. Let C_1' and C_2' be the λ-approximate clusterings for J_1 and J_2 respectively. Hence from Lemma 5 at any later time $(t' + W - 1)$, such that $J_1 \leq t' < J_2$, $Cost([t, t+W-1], C_1') \leq Cost([J_1, t+W-1], C_1') \leq \lambda \cdot OPT([J_1, t+W-1]) \leq \lambda(2 + \gamma^2\lambda) \cdot OPT([I_{i+1}, t + W - 1]) \leq \lambda(2 + \gamma^2\lambda) \cdot OPT([t, t + W - 1])$ (Using monotonicity of OPT and smoothness). So, the final approximation is $(2 + \gamma^2\lambda)$. We find subsequent inner indices in similar manner. Note that, after crossing each inner index, $\tilde{n}_{ij}$ for some j reduces by a factor $\frac{18}{11}$. Since there are at most k medians and W items, the total number of inner indices between I_1 and I_2 is at most $O(k \log W)$. Also note that, for checking Equation 2, the coordinator needs $\tilde{n}_{ij}^{[t,aW+W]}$ values, for various values of t, and j, which we obtain as follows.

Each node makes a backward pass over its local data. During this pass, for each i, it maps each point $p \in [I_i, aW + W]$ to c_{ij^*}, where $j^* = \arg\min_j d(p, c_{ij})$, i.e. c_{ij^*} is the closest of the medians from C_i. It also keeps a counter n_{ij} for each c_{ij}, which increments for each new point mapping to c_{ij}. We then record the time-points where n_{ij} increases by $(1 + \frac{1}{20})$-factor, i.e crosses $\{\frac{21}{20}, \frac{21}{20}^2, \ldots, W\}$ for the first time. We call this set of time-points as H_{ij}. Each node z sends such H_{ij}^z, for each i, j to the coordinator. Note that, $n_{ij}^{[t,aW+W]} = \sum_z n_{ij}^{[t,aW+W]_z}$, where $[t, aW + W]_z$ denotes items from $[t, aW + W]$ that appear at node z. Using H_{ij}^z, coordinator can approximate $n_{ij}^{[t,aW+W]_z}$ upto $(1 \pm \frac{1}{10})$ factor, for any z. Taking sum over all z, it can approximate $n_{ij}^{[t,aW+W]}$ for any t, upto $(1 \pm \frac{1}{10})$ factor, as required.

The total number of indices are $O(kL \log W)$. The backward online algorithm costs $\langle O(km \log^3 W), O(km \log W)), O(k \log W) \rangle$. Communicating the H_{ij}^z values cost $O(kmL$ $\log^3 W)$ in total. The set of inner indices require $O(kL \log^2 W)$ computations for n_{ij}^S per W items by the coordinator, and we ignore the costs for these. Later, online algorithms are run for each index for at most W arrivals. We also have to include $O(W)$ space complexity for storing the static windows. The final cost is $\langle O(k^2 mL\log^4 W), O(k^2 mL\log^2 W + W), O(k^2 L\log^2 W)\rangle$. Since we run at most $O(kL\log W)$ online algorithms, success probability per W items is still $(1 - \frac{1}{\mathrm{poly}(W)})$. ◀

Next we present a Lemma analogous to Lemma 5 for smoothness of k-center clustering. The additional property is more relaxed than that of k-median. To the best of our knowledge, this result was not known before. We denote by $Cost(P, O)$ the k-center clustering cost for a set of points P, when O is the set of k centers. For convenience, we drop the k in OPT_k when there is no ambiguity.

▶ **Lemma.** *For any distinct sets of points $A, B, C \subseteq \chi$ from some metric space χ, $\mathrm{OPT}(A \cup B) \leq \gamma \mathrm{OPT}(B) \implies \mathrm{OPT}(A\cup B\cup C) \leq (1+2r\gamma)\mathrm{OPT}(B\cup C)$ for any $r, \gamma \geq 1$, provided the following property holds for the sets A, B: There is a k-center clustering $t : (A\cup B) \to F$, upto approximation ratio r, such that, for each center $f \in F, |t^{-1}(f)\cap A| > 0 \implies |t^{-1}(f)\cap B| > 0$.*

Proof. Let O be the optimal set of centers for $B\cup C$. We will map each element $a \in A$ to some point in O. Let O' be the r-approximate set of centers for $A \cup B$, which satisfies the above property. Let $o' \in O'$ be the center to which $a \in A$ maps to. By assumption some $b \in B$ also gets mapped to o'. Finally, let $o \in O$ be the center to which b maps to. We will map a to o. Then by definition, $\max(d(a, o'), d(b, o')) \leq r \cdot \mathrm{OPT}(A\cup B)$ and $d(b, o) \leq \mathrm{OPT}(B\cup C)$. Then, by triangle inequality, $d(a, o) \leq (d(a, o')+d(b, o')+d(b, o)) \leq (2r\cdot\mathrm{OPT}(A\cup B)+\mathrm{OPT}(B\cup C))$

$$\begin{aligned}
OPT(A \cup B \cup C) &\leq Cost(A \cup B \cup C, O)\\
&\leq \max\{\mathrm{OPT}(B \cup C), Cost(A, O)\}\\
&\leq \max\{\mathrm{OPT}(B \cup C), (2r \cdot \mathrm{OPT}(A \cup B) + \mathrm{OPT}(B \cup C))\}\\
&\leq (2r \cdot \mathrm{OPT}(A \cup B) + \mathrm{OPT}(B \cup C))\\
&\leq (2r\gamma \cdot \mathrm{OPT}(B) + \mathrm{OPT}(B \cup C)) && \text{(Given)}\\
&\leq (1 + 2r\gamma)\mathrm{OPT}(B \cup C) && \text{(Using monotonicity of OPT)}
\end{aligned}$$

◀

Our dsw algorithm for k-center clustering closely follows that for k-median clustering given earlier in this section. We create a set of 'outer' indices corresponding to a constant factor drop of the cost of the diw algorithm. We also introduce a set of 'inner' indices between each pair of outer indices to satisfy the additional property of Lemma 5. These inner indices are created at a point t, such that there exists a center to which no item from part $(t, aW + W]$ maps. Since, there are at most k centers, at most k inner indices are possible between each pair of outer indices. Hence the total number of indices is $O(k\log W)$. We skip more details of the proof since it closely follows that of Theorem 15.

References

1 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the Twenty-Second Symposium on Principles of Database Systems PODS*, pages 234–243, 2003.

2 Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 28–39, 2003.

3 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering on sliding windows in polylogarithmic space. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 350–364, 2015.

4 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1374–1390, 2016.

5 Vladimir Braverman and Rafail Ostrovsky. Effective computations on sliding windows. *SIAM J. Comput.*, 39(6):2113–2131, 2010.

6 Ho-Leung Chan, Tak Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. Continuous monitoring of distributed data streams over a time-based sliding window. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS*, pages 179–190, 2010.

7 Jiecao Chen and Qin Zhang. Improved algorithms for distributed entropy monitoring. *Algorithmica*, 78(3):1041–1066, Jul 2017.

8 Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, pages 19:1–19:12, 2016.

9 Graham Cormode. Algorithms for continuous distributing monitoring: A survey. In *First International Workshop on Algorithms and Models for Distributed Event Processing*, pages 1–10, 2011.

10 Graham Cormode, Minos N. Garofalakis, S. Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 25–36, 2005.

11 Graham Cormode, S. Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1076–1085, 2008.

12 Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the Twenty-Ninth ACM Symposium on Principles of Database Systems, PODS*, pages 77–86, 2010.

13 Graham Cormode, S. Muthukrishnan, and Wei Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE*, pages 1036–1045, 2007.

14 Graham Cormode and Ke Yi. Tracking distributed aggregates over time-based sliding windows. In *Scientific and Statistical Database Management - 24th International Conference, SSDBM*, pages 416–430, 2012.

15 Abhinandan Das, Sumit Ganguly, Minos N. Garofalakis, and Rajeev Rastogi. Distributed set expression cardinality estimation. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 312–323, 2004.

16 Sutanu Gayen and N. V. Vinodchandran. Algorithms for k-median clustering over distributed streams. In *Computing and Combinatorics - 22nd International Conference, COCOON*, pages 535–546, 2016.

17 Zengfeng Huang, Ke Yi, and Qin Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 295–306, 2012.

18 Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 289–300, 2006.

19 Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 563–574, 2003.

20 Nicolo Rivetti, Yann Busnel, and Achour Mostéfaoui. Efficiently summarizing data streams over sliding windows. In *14th IEEE International Symposium on Network Computing and Applications, NCA*, pages 151–158, 2015.

21 Srikanta Tirthapura and David P. Woodruff. Optimal random sampling from distributed streams revisited. In *Distributed Computing - 25th International Symposium, DISC*, pages 283–297, 2011.

22 David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 941–960, 2012.