Sliding Window Computations over Data Streams

Brian Babcock[*] Mayur Datar[†] Rajeev Motwani[‡] Liadan O'Callaghan[§]

Abstract

The sliding window model, where data elements arrive continually and only the most recent N elements are used when answering queries, has been deemed appropriate for most applications that handle data in a streaming fashion. We present a novel technique for solving two important and related problems in the sliding window model — maintaining variance and maintaining a k–medians clustering. Our solution to the problem of maintaining variance provides a continually updated estimate of the variance of the last N values in a data stream with relative error of at most ϵ using $O(\frac{1}{\epsilon^2}\log N)$ memory. We present a constant-factor bicriteria approximation algorithm which maintains approximate k-medians for the last N data points using $O(\frac{k}{\tau^4}N^{2\tau}\log^2 N)$ memory, where $\tau < 1/2$ is a parameter which trades off the space bound with the approximation factor of $O(2^{O(1/\tau)})$. We also present an algorithm that uses exactly k centers, as opposed to $2k$ in the previous result, and has the same approximation guarantee of $O(2^{O(1/\tau)})$; however, this algorithm is more complicated and requires $O(\frac{k^{1/\tau}}{\tau^4}N^{2\tau}\log^2 N)$ memory.

[*]Department of Computer Science, Stanford University, Stanford, CA 94305. Email: `babcock@cs.stanford.edu`. Supported in part by a Stanford Graduate Fellowship and NSF Grant IIS-0118173.

[†]Department of Computer Science, Stanford University, Stanford, CA 94305. Email: `datar@cs.stanford.edu`. Supported in part by a Microsoft Graduate Fellowship and NSF Grant IIS-0118173.

[‡]Department of Computer Science, Stanford University, Stanford, CA 94305. Email: `rajeev@cs.stanford.edu`. Supported in part by NSF Grant IIS-0118173, an Okawa Foundation Research Grant, and Veritas.

[§]Department of Computer Science, Stanford University, Stanford, CA 94305. Email: `loc@cs.stanford.edu`. Research supported in part by NSF Graduate Fellowship and NSF Grant IIS-0118173.

1 Introduction

A new model of computation called *data streams* has emerged in the database community [4], motivated by applications involving massive data sets (much larger than available main memory) that need to be processed in a single pass. The data stream model, in which data items are presented as a possibly infinite sequence of records, captures many of the required characteristics of such applications, as distinct from standard applications that can be processed in random-access memory. In telecommunications, for example, call records are generated continuously; the information they contain cannot reasonably be extracted except by a small-memory algorithm that operates in one pass. The collection and analysis of the streams of packet headers from routers can aid network traffic engineers. There are many other traditional and emerging applications in which data streams play an important and natural role, such as web tracking/personalization, network security, medical monitoring, sensor databases, and finance. There are also applications such as data mining and OLAP in which stored data is treated as a stream because the volume of data on disk is so large that it is only possible to make one pass. Refer to the survey [4] for more details on applications and prior work.

A challenging issue in processing data streams is that although they are of unbounded length, making storing the entire stream impossible, for many applications it is still important to retain an ability to execute queries referencing past data. Given memory constraints, it becomes necessary to devise techniques for maintaining summaries or synopses of the stream's history. However, in most applications very old data is considered less useful than more recent data. Data from the recent past is more likely to be relevant and interesting than older data. The *sliding window* model [4, 9] has emerged as a mechanism for discounting stale data. Here, only the last N elements to arrive in the stream are considered relevant for answering queries, where N is the *window size*.

In a recent paper, Datar, Gionis, Indyk, and Motwani [9] considered the problem of maintaining simple statistics over sliding windows, which is an important tool for processing data stream queries. They presented a general framework based on a novel data structure called an *exponential histogram (EH)* to estimate a class of aggregate functions over sliding windows. Their result applies to any function f satisfying the following properties for all multisets X, Y: **(1)** $f(X) \geq 0$; **(2)** $f(X) \leq \mathit{poly}(|X|)$; **(3)** $f(X \cup Y) \geq f(X) + f(Y)$; and, **(4)** $f(X \cup Y) \leq C_f(f(X) + f(Y))$, where $C_f \geq 1$ is a constant. Furthermore, for their technique to be efficient, it must be possible to compute (or at least estimate) $f(X)$ from a small *sketch* of X, where a sketch is a synopsis structure of small size which is *additive* — given the sketches for two multisets X and Y, a sketch for $X \cup Y$ can be quickly computed. They observe that *sum* and l_1/l_2 vector norms satisfy these properties. However, many interesting statistics do not satisfy these properties and cannot be estimated using their techniques.

In this paper, we build upon the EH technique, extending it to work for statistics that do not obey the above properties. We consider two important statistics — variance and k-medians. Note that the variance is nothing but the k-medians cost under the *sum of squares* metric for $k = 1$. The technique that we use to solve both problems is to summarize intervals of the data stream using combinable synopses. In order to make efficient use of memory, synopses for adjacent intervals are combined when doing so will not increase the relative error significantly. The synopsis for the interval that straddles the sliding window boundary is inaccurate because some of the points it summarizes no longer fall within the sliding window, but we are able to contain this inaccuracy by appropriately adjusting the synopsis, treating the expired points as though they were "typical" points from the interval.

Our main results are as follows. We show how to estimate variance over sliding windows with relative error at most ϵ using $O(\frac{1}{\epsilon^2} \log N)$ memory. Further, we present an algorithm for k–medians clustering over sliding windows using $O(\frac{k}{\tau^4} N^{2\tau} \log^2 N)$ memory. This is a constant-factor bicriteria approximation algorithm for the k–median problem — it uses $2k$ centers and the objective function value is within a constant factor ($2^{O(1/\tau)}$) of optimal, where $\tau < 1/2$ is a parameter which captures the trade-off between the space bound and the approximation ratio. We build upon this solution to devise an algorithm using *exactly* k centers and providing the same approximation guarantee of $(2^{O(1/\tau)})$, albeit at the cost of an increased memory requirement of $O(\frac{k^{1/\tau}}{\tau^4}N^{2\tau}\log^2 N)$.

Our result for clustering adapts the one-pass algorithm for clustering data streams of Guha, Mishra, Motwani, and O'Callaghan [17]. Their algorithm uses $O(n^\tau)$ memory and provides a constant factor $(2^{O(1/\tau)})$ approximation to the k–medians problem. Our adaptation to the sliding window model also makes use of EH to estimate the value of the k–medians objective function, though direct application of the techniques from Datar, Gionis, Indyk, and Motwani [9] is impossible (as in variance) because the k–medians objective function does not satisfy property (4) from above. If there are two sets of data points, each tightly clustered about its mean, then the variance of each set will be small; however, if the means of the two sets are far apart, then the variance of their union can be arbitrarily large. The same is true for clustering — although each of two sets of points may cluster nicely into k clusters, if the two sets of cluster centers are far apart, the union of the two sets is difficult to cluster.

The violated property (4) is crucial to the EH technique. A major issue in the sliding window model is figuring out how to eliminate expired data; since there is not enough memory to store the entire window, at best it is only possible to have a rough idea of the value of the expired data elements. Exponential histograms deal with this problem by grouping data elements into buckets covering contiguous time intervals and maintaining summary statistics for each bucket. The statistics for the bucket straddling the sliding window boundary will be inaccurate because it contains expired data and there is no way to distinguish the relative contribution of such stale data towards the overall bucket statistics. The EH algorithm ensures that no bucket greatly influences the function being computed; therefore, while estimating the function, the error due to the oldest bucket containing expired data can be suitably bounded. However, when a single bucket may greatly influence the value of the function (as in variance and k–medians), the EH technique no longer works.

1.1 Related Work

Algorithms for streaming data have been an area of much recent research interest. Early work [1, 12, 18] on stream computations addresses the problems of approximating frequency moments and computing the l_p differences of streams. More recent work [13, 14, 15, 16] has focused on building summaries like histograms and wavelet coefficients over data streams. Domingos et al. [10, 11] have studied the problem of maintaining decision trees over data streams. As mentioned earlier, Guha et al. [17] studied the problem of one-pass clustering of data streams. Previous work on k–median clustering includes [2, 3, 6, 7, 8, 19, 20, 22, 21, 23, 24]. Datar et al. [9] have considered the problem of maintaining simple statistics over sliding windows. Our work can be considered an extension of that work; we estimate functions over sliding windows that cannot be estimated using their techniques. A detailed survey of the algorithmic and database research in data streams is also available [4].

1.2 Model and Summary of Results

In the *sliding window model*, data elements arrive in a stream and only the last N (window size) elements to have arrived are considered relevant at any moment. These most recent N elements are called *active* data elements; the rest are termed *expired* and they no longer contribute to query answers or statistics on the data set. Once a data element has been processed, it cannot be retrieved for further computation at a later time, unless it is explicitly stored in memory. The amount of memory available is assumed to be limited, in particular, sublinear in the size of the sliding window. Therefore algorithms that require storing the entire set of active elements are not acceptable within this model.

We employ the notion of a *timestamp*, which corresponds to the position of an active data element in the current window. We timestamp the active data elements from most recent to oldest with the most recently arrived data element having a timestamp of 1. Let x_i denote the data element with timestamp i. Clearly, the timestamps change with every new arrival, and we do not wish to make explicit updates. A simple solution is to record the arrival times in a wraparound counter of $\log N$ bits; then the timestamp can be extracted by comparison with the counter value of the current arrival.

We will maintain histograms for the active data elements in the data stream. Our notion of histograms is far more general than the traditional one used in the database literature. In particular, every bucket in our histograms stores some summary/synopsis structure for a contiguous set of data elements, i.e., the histogram is partitioned into buckets based on the arrival time of the data elements. Along with this synopsis, for every bucket, we keep the timestamp of the most recent data element in that bucket (the *bucket timestamp*). When the timestamp of a bucket reaches $N+1$, all data elements in the bucket have expired, so we can drop that bucket and reclaim its memory. All buckets, save the last, contain only active elements, while the last bucket may contain some expired elements besides at least one active element. The buckets are numbered $B_1, B_2, \ldots, B_m$, starting from most recent (B_1) to oldest (B_m); further, $t_1, t_2, \ldots, t_m$ denote the bucket timestamps.

We now formally state the problems considered and our results.

Problem 1 *Given a stream of numbers, maintain at every instant the variance of the last N values, $VAR = \sum_{i=1}^{N}(x_i - \mu)^2$, where $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ denotes the mean of the last N values.*

Unless we decide to buffer the entire sliding window in memory, we cannot hope to compute the variance exactly at every instant. In Section 2 we present a small-space algorithm to solve this problem approximately. Our algorithm uses $O(\frac{1}{\epsilon^2}\log N)$ memory and provides an estimate at every instant that has relative error at most ϵ. The time required per new element is amortized $O(1)$.

We then extend our work to a clustering problem. Given a multiset X of objects in a metric space M with distance function d, the *k–medians problem* is to choose k points $c_1, \ldots, c_k \in M$ so as to minimize $\sum_{x \in X} d(x, C(x))$, where $C(x)$ is the closest of $c_1, \ldots, c_k$ to x.[1] If $c_i = C(x)$ then x is said to be *assigned* to c_i, and $d(x, c_i)$ is called the *assignment distance* of x. The objective function is the sum of assignment distances.

Problem 2 (SWKM) *Given a stream of points from a metric space M with distance function d, window size N, and parameter k, maintain at every instant t a median set $c_1, c_2, \ldots, c_k \in M$ minimizing $\sum_{x \in X_t} d(x, C(x))$, where X_t is the multiset of the N most recent points at time t.*

In Section 3 we show how to maintain approximate k–medians over sliding windows using $\tilde{O}(\frac{k}{\tau^4}N^{2\tau})$ memory, for any $\tau < 1/2$, using amortized $\tilde{O}(k)$ insertion time per data point. The algorithm produces $2k$ medians such that the sum of assignment distances of the points in X_t is within a constant multiplicative factor of $2^{O(1/\tau)}$ of the assignment cost attainable by an optimal choice of k medians. We remove the need for a bicriteria relaxation in Section 3.4 using a more sophisticated algorithm which produces exactly k clusters, with a slightly increased memory requirement of $\tilde{O}(\frac{k^{1/\tau}}{\tau^4}N^{2\tau})$.

2 Maintaining Variance over Sliding Windows

Datar, Gionis, Indyk, and Motwani [9] presented a space lower bound of $\Omega(\frac{1}{\epsilon}\log N(\log N + \log R))$ bits for approximately (with error at most ϵ) maintaining the sum, where N is the sliding window size and each data value is at most R. Assuming $R = poly(N)$, this translates to a lower bound of $\Omega(\frac{1}{\epsilon}\log N)$ words for maintaining the sum, and hence the variance, of the last N elements. There is a gap of $\frac{1}{\epsilon}$ between this lower bound and the upper bound obtained here.

We now describe our algorithm for solving Problem 1, i.e., to compute an estimate of the variance with relative error at most ϵ. As mentioned before, elements in the data stream are partitioned into

[1]This formulation of k–medians is called *continuous k–medians*. In *discrete k–medians*, the medians must be chosen from the set X. For us, "k–medians" will refer to the continuous version.

buckets by the algorithm. For each bucket B_i, besides maintaining the timestamp t_i of the most recent data element in that bucket, our algorithm maintains the following summary information: the number of elements in the bucket (n_i), the mean of the elements in the bucket (μ_i), and the variance of the elements in the bucket (V_i). The actual data elements that are assigned to a bucket are not stored.

In addition to the buckets maintained by the algorithm, we define another set of *suffix buckets*, denoted $B_{1^*}, \ldots, B_{j^*}$, that represent suffixes of the data stream. Bucket B_{i^*} represents all elements in the data stream that arrived after the elements of bucket B_i, that is, $B_{i^*} = \bigcup_{l=1}^{i-1} B_l$. Except for the bucket B_{m^*}, which represents all points arriving after the oldest non-expired bucket, these suffix buckets are not maintained by the algorithm, though their statistics are calculated temporarily during the course of the algorithm described below.

Algorithm Insert: x_t denotes the latest element.

1. If $x_t = \mu_1$, then extend bucket B_1 to include x_t, by incrementing n_1 by 1. Otherwise, create a new bucket for x_t. The new bucket becomes B_1 with $V_1 = 0$, $\mu_1 = x_t$, $n_1 = 1$. An old bucket B_i becomes B_{i+1}.
2. If the last bucket B_m has timestamp greater than N, delete the bucket. Bucket B_{m-1} becomes the new "oldest" bucket. Maintain the statistics of B^*_{m-1} (instead of B^*_m), which can be computed using the previously maintained statistics for B^*_m and statistics for B_{m-1}.
3. Let $k = \frac{9}{\epsilon^2}$ and $V_{i,i-1}$ denote the variance of the bucket obtained by combining buckets B_i and B_{i-1}. While there exists an index $i > 2$ such that $kV_{i,i-1} \leq V^*_{i-1}$, find the smallest such i and combine buckets B_i and B_{i-1} using the combination rule described below. Note that the statistics for B^*_i can be computed incrementally from the statistics for B_{i-1} and B_{i-1^*}.
4. Output estimated variance at time t according to the estimation procedure below.

Combination Rule: While maintaining the histogram, our algorithm sometimes needs to combine two adjacent buckets. The statistics (count, mean, and variance) for the resulting combined bucket are computed according to the following combination rule. Consider two buckets B_i and B_j that get combined to form a new bucket $B_{i,j}$. The statistics for $B_{i,j}$ are computed from the statistics of B_i and B_j as follows: $n_{i,j} = n_i + n_j$; $\mu_{i,j} = \frac{\mu_i n_i + \mu_j n_j}{n_{i,j}}$; and, $V_{i,j} = V_i + V_j + \frac{n_i n_j}{n_{i,j}}(\mu_i - \mu_j)^2$. Note that the combination rule can also be used to "delete" a set of points (A) from a larger set ($B \supseteq A$), i.e., calculate the statistics corresponding to the difference ($B - A$), based on the statistics for the two sets A, B. The following lemma (whose proof is in the Appendix) shows the correctness of the statistics computed by the combination rule.

Lemma 1 *The bucket combination procedure correctly computes $n_{i,j}$, $\mu_{i,j}$, and $V_{i,j}$ for the new bucket.*

Estimation: Let $B_1, \ldots, B_m$ be the set of histogram buckets at time t. We describe a procedure to estimate the variance over the current active window. Let B_m be the earliest active bucket. It contains some active elements, including the one with timestamp N, but may also contain expired data. We maintain statistics corresponding to B_{m^*}, the suffix bucket containing all elements that arrived after bucket B_m. To this end, we use the combination rule for every new data element that arrives. Whenever the earliest bucket gets deleted, we can find the new B_{m^*} by "deleting" the contribution of the new earliest bucket (B_m) from the previous B_{m^*}, using the combination rule.

Let $B_{\tilde{m}}$ refer to the non-expired portion of the bucket B_m, i.e., the set of elements in B_m that are still active. (See Figure 1 in the Appendix for an illustration.) Since we do not know the statistics $n_{\tilde{m}}$, $\mu_{\tilde{m}}$, and $V_{\tilde{m}}$ corresponding to bucket $B_{\tilde{m}}$, we estimate them as follows: $n^{EST}_{\tilde{m}} = N + 1 - t_m$; $\mu^{EST}_{\tilde{m}} = \mu_m$; and, $V^{EST}_{\tilde{m}} = \frac{V_m}{2}$. Note that $n_{\tilde{m}}$ is exact, i.e., $n^{EST}_{\tilde{m}} = n_{\tilde{m}}$.

The statistics for $B_{\tilde{m}}$ and B_{m^*} are sufficient to accurately compute the variance at the time instant t. In fact the variance is nothing but the variance corresponding to the bucket $B_{\tilde{m},m^*}$ obtained by combining $B_{\tilde{m}}$ and B^*_m. Therefore, by the combination rule, the actual variance ($\hat{\text{VAR}}(t)$) for the current active window is given by: $\hat{\text{VAR}}(t) = \left(V_{\tilde{m}} + V_{m^*} + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(\mu_{\tilde{m}} - \mu_{m^*})^2\right)$. At every time instant t we estimate the variance by computing the variance of $B_{\tilde{m},m^*}$ using the estimates above for $B_{\tilde{m}}$. This estimate can be found in $O(1)$ time provided we maintain the statistics for $B_{\tilde{m}}$ and B^*_m. The error in our estimate arises due to the error in the estimate of the statistics for $B_{\tilde{m}}$. As we shall prove below, this error is small (within factor ϵ) as compared to the exact answer ($\hat{\text{VAR}}(t)$) provided we maintain the following invariant:

Invariant 1 *For every bucket B_i, $kV_i \leq V_{i^*}$, where $k = \frac{9}{\epsilon^2}$.*

The algorithm maintains an additional invariant ensuring that the total number of buckets is small:

Invariant 2 *For every bucket B_i ($i > 1$), $kV_{i,i-1} > V_{i-1^*}$, where $k = \frac{9}{\epsilon^2}$.*

The proof of the next lemma is omitted.

Lemma 2 *The number of buckets maintained at any point in time by an algorithm that maintains Invariant 2 is $O(\frac{1}{\epsilon^2}\log NR^2)$, where R is an upper bound on the absolute value of the data elements.*

Note that we require $\Omega(\log R)$ bits of memory to represent each data element. The proof of the following technical lemma is in the Appendix.

Lemma 3 *For any choice of a and any set of data elements B with mean μ, $\sum_{x\in B}(x-a)^2 = \sum_{x\in B}(x-\mu)^2 + |B|(a-\mu)^2$*

The following theorem summarizes the algorithm's performance; the proof is in the Appendix.

Theorem 1 *Let $VAR(t)$ be the variance estimate provided by the algorithm maintaining Invariants 1 and 2, and let $\hat{VAR}(t)$ be the actual variance. Then $(1-\epsilon)\hat{VAR}(t) \leq VAR(t) \leq (1+\epsilon)\hat{VAR}(t)$. Further, this algorithm uses $O(\frac{1}{\epsilon}\log NR^2)$ memory.*

The algorithm presented earlier requires $O(\frac{1}{\epsilon^2}\log NR^2)$ time per new element. Most of the time is spent in Step 3 where we make the sweep to combine buckets. This time is proportional to the size of the histogram ($O(\frac{1}{\epsilon^2}\log NR^2)$). A simple trick is to skip Step 3 till we have seen $\Theta(\frac{1}{\epsilon^2}\log NR^2)$ data points. This ensures that the running time of the algorithm is amortized $O(1)$. While we may violate Invariant 2 temporarily, we restore it every $\Theta(\frac{1}{\epsilon^2}\log NR^2)$ data points when we execute Step 3. This ensures that the number of buckets is $O(\frac{1}{\epsilon^2}\log NR^2)$.

3 Clustering over Sliding Windows

We now present a solution to the SWKM problem (Problem 2). As mentioned earlier, we focus on continuous k–medians; however, the techniques also apply to discrete k–medians, although the approximation ratios will be different. Our solution is an adaptation to the sliding window model of the algorithm of Guha, Mishra, Motwani, and O'Callaghan [17]. We begin by briefly reviewing some important ideas from there and giving some intuition that will be useful in explaining our solution.

3.1 Memory-Constrained k-Medians Algorithm

Guha, Mishra, Motwani, and O'Callaghan [17] presented a small-space, constant-factor approximation algorithm for clustering in data streams. This is a divide-and-conquer algorithm which can be viewed as incremental hierarchical agglomerative clustering (HAC). Let n be the number of input points. On seeing the first m points, reduce them to $O(k)$ points using any bicriteria clustering algorithm A that generates $O(k)$ medians and gives a solution of cost within a constant factor of optimal k–medians cost; for example, the linear-space algorithm of Charikar and Guha [6] produces at most $2k$ medians within a constant factor of optimal. We discard the original m points and store only the $O(k)$ medians (called "level-1 medians") along with weights denoting the number of points assigned to these medians. Repeating this $m/O(k)$ times yields m medians with associated weights. We cluster these weighted medians to obtain $O(k)$ level-2 medians. In general, we maintain at most m level-i medians and on seeing m of them, generate $O(k)$ level-$(i+1)$ medians of weight equal to the sum of the weights of the intermediate medians assigned to them. For $m = O(n^\tau)$, with constant $\tau < 1$, we obtain a constant factor ($2^{O(1/\tau)}$) approximation algorithm requiring $O(n^\tau)$ memory. We view the solution as a tree with leaf nodes as data elements and interior nodes as medians. Each node's parent is the median of the next higher level to which it is assigned. A node's weight is the number of leaves in the subtree rooted at it. At all times, the memory contents maintain a frontier of the tree with nodes from different levels. (See Figure E in the Appendix.)

The following theorems due to Guha, Mishra, Motwani, and O'Callaghan [17] are key to the analysis of our divide-and-conquer strategy. The first theorem states that the sum of the individual costs of optimum solutions for the different partitions is at most the cost of the optimum solution for the entire set. In the algorithm described above these intermediate medians are further clustered when enough of them are gathered. The second theorem bounds the cost of an optimum solution for the instance (χ') that consists of $O(lk)$ weighted medians, where the weight of every median is the number of points assigned to it in the solution corresponding to its partition.

Theorem 2 (Theorem 2.2 [17]) *Consider any set of n points arbitrarily partitioned into disjoint sets $\chi_1, \dots, \chi_l$. The sum of optimum solution values for the k–medians problem on the l sets of points is at most the cost of the optimum k–medians solution for all the n points.*

Theorem 3 (Theorem 2.3 [17]) *If C is the sum of the costs of the l optimum k–medians solutions for $\chi_1, \dots, \chi_l$ and C^* is the cost of the optimum k–medians solution for the entire set S, then there exists a solution of cost at most $C + C^*$ to the new weighted instance χ'.*

Using these two theorems it is not difficult to prove (see Theorem 2.4 in [17]) that if we have a bicriteria b-approximation algorithm (where at most ak medians are output with cost at most b times the optimum k–medians solution) for clustering the l partitions and a c approximation algorithm for clustering the weighted medians we get an $(c(1+b)+b)$-approximation algorithm. Guha et al. apply this idea recursively to get a $2^{O(1/\tau)}$ factor approximation algorithm using $O(n^\tau)$ memory. The overall running time of their algorithm is $\tilde{O}(nk)$. It is important to note that the approximation factor increases exponentially with the depth of the hierarchical tree.

3.2 Bicriteria k-Medians over Sliding Windows

We now adapt the above algorithm to sliding windows using ideas from the previous section. This section is organized as follows: first, we describe the EH data structure used by our algorithm and the method for associating a cost with each bucket; next, we describe the combination step for combining two buckets; then, we discuss the estimation procedure of the final clustering of the current active window after each data point arrives, or whenever a clustering is desired; and finally, we present the overall algorithm that maintains the data structure as new points arrive.

Data Structure: The data structure maintained is an EH whose buckets are numbered $B_1, B_2, \dots, B_m$ from most recent to oldest, and buckets containing only expired points are discarded. As with variance, each bucket stores a summary structure for a set of contiguous points as well as the timestamp of the most recent point in the bucket. In the case of variance, the summary structure contained the triplet (n_i, μ_i, V_i); for clustering, each bucket consists of a collection of data points or intermediate medians. For consistency, we will refer to the original points as level-0 medians.

Each median is represented by the triple $(p(x), w(x), c(x))$. The value $p(x)$ is the identifier of x; in Euclidean space, for example, this could be the coordinates of x, and in a general metric this could simply be the index of x in the point set. The value $w(x)$ is the weight of a median x, i.e., the number of points that x represents. If x is of level 0, $w(x) = 1$, and if x is of level i, then $w(x)$ is the sum of the weights of the level-$(i-1)$ points that were assigned to x when the level-i clustering was performed.

Finally, $c(x)$ is the estimated cost of x, i.e., an estimate of the sum of the costs $d(x, y)$ of assigning to x each of the leaves y of the subtree rooted at x. If x is of level 0, $c(x) = 0$; if x is of level 1, $c(x)$ is the sum of assignment distances of the members of x; and, if x is of level $i > 1$, $c(x)$ is the sum over all members y of x, of $c(y) + w(y) \cdot d(x, y)$. Thus, $c(x)$ is an overestimate of the "true" cost of x.

As in the algorithm of Guha et al. [17], we maintain medians at intermediate levels and whenever there are N^τ medians at the same level we cluster them into $O(k)$ medians at the next higher level. Thus, each bucket B_i can be split into $1/\tau$ (the maximum tree height) groups $R_i^0, \ldots, R_i^{1/\tau-1}$, where each R_i^j contains medians at level j. Each group contains at most N^τ medians. Along with a collection of medians, the algorithm also maintains for each bucket B_i a *bucket cost*, which is a conservative overestimate of the assignment cost of clustering all the points represented by the bucket.

Cost Function: The bucket cost function is an estimate of the assignment cost of clustering the points represented by the bucket. Consider a bucket B_i and let $Y_i = \bigcup_{j=0}^{1/\tau-1} R_i^j$ be the set of medians in the bucket. As explained earlier, each median x is associated with a cost $c(x)$ which is an estimate of the sum of the distances to x from all points assigned to x by the clustering algorithm. We cluster the points in Y_i to produce k medians $c_1, \ldots, c_k$. The cost function for bucket B_i is given by: $f(B_i) = \sum_{x \in Y_i} c(x) + w(x) \cdot d(x, C(x))$, where $C(x) \in \{c_1, \ldots, c_k\}$ is the median closest to x.

Combination: Let B_i and B_j be two (typically adjacent) buckets that need to be combined to form the merged bucket $B_{i,j}$, and let $R_i^0, \ldots, R_i^{1/\tau-1}$ and $R_j^0, \ldots, R_j^{1/\tau-1}$ be the groups of medians from the two buckets, where R_i^l (similarly, R_j^l) represents the group of medians at level l in bucket B_i. We set $R_{i,j}^0 = R_i^0 \bigcup R_j^0$. If $|R_{i,j}^0| > N^\tau$, then cluster the points from $R_{i,j}^0$ and set $R_{i,j}^0$ to be empty. Let C_0 denote the set of $O(k)$ medians obtained by clustering $R_{i,j}^0$, or the empty set if $R_{i,j}^0$ did not need to be clustered. We carry over these level-1 medians to the next group $R_{i,j}^1$. Set $R_{i,j}^1 = R_i^1 \bigcup R_j^1 \bigcup C_0$. As before, if there are more than N^τ medians, we cluster them to get a carry-over set C_1, and so on. In general, after at most $1/\tau$ unions, each possibly followed by clustering, we get the combined bucket $B_{i,j}$. Finally, the bucket cost is computed by clustering all medians in the bucket (at all levels).

Estimation: Let $B_1, B_2, \ldots, B_m$ denote the buckets at any time instant t. B_m is the last bucket and contains medians that represent some data points that have already expired. If a query is posed at this moment that asks for a clustering of the active elements we do the following:

- Consider all but the last of the buckets: $B_1, \ldots, B_{m-1}$. They each contain at most $\frac{1}{\tau}N^\tau$ medians. We will prove that the number of buckets is $O(\log N)$. Thus we have $O(\frac{1}{\tau}N^\tau \log N)$ medians. Cluster them to produce k medians.
- Similarly, cluster bucket B_m to produce k additional medians.
- Present the $2k$ medians as the answer.

If required, the procedure can also provide an estimate for the assignment cost using the same technique as that used for computing the cost function over buckets.

Algorithm: The algorithm for combining and maintaining buckets is very similar to that used for estimating variance. As before, we define suffix buckets B_{i^*} which represent the combination of all buckets that are later than a particular bucket (B_i). These are not maintained at all times but instead are computed when required, as in the case of variance.

Algorithm Insert: x_t denotes the latest element.

1. If there are less than k level-0 medians in B_1 add the point x_t as a level-0 median in bucket B_1. Otherwise, create a new bucket B_1 to contain x_t and renumber the existing buckets accordingly.
2. If bucket B_m has timestamp more than N, delete it; now, B_{m-1} becomes the last bucket.
3. Make a sweep over the buckets from most recent to least recent and while there exists an index $i > 2$ such that $f(B_{i,i-1}) \leq 2f(B_{i-1^*})$, find the smallest such i and combine buckets B_i and B_{i-1} using the combination procedure described above. The suffix bucket B_{i^*} is computed as we make the sweep.

Our algorithm maintains the following two invariants, which leads to the following two lemmas.

Invariant 3 *For every bucket B_i, $f(B_i) \leq 2f(B_{i^*})$.*

Invariant 4 *For every bucket B_i ($i > 1$), $f(B_{i,i-1}) > 2f(B_{i-1^*})$.*

Lemma 4 *The algorithm, which maintains Invariant 3, produces a solution with $2k$ medians whose cost is within a multiplicative factor of $2^{O(1/\tau)}$ of the cost of the optimal k–medians solution.*

Proof Sketch: Let B_m be the last bucket in the histogram. Recall that B_m may contain some medians that represent points that have expired. Consider first the suffix bucket B_{m^*} representing all points from buckets with timestamps later than B_m and compare our algorithm's performance on this set of points to the optimal solution on the same set of points.

We cluster medians at any level only when there are at least N^τ medians at that level, which guarantees that the depth of the hierarchical clustering tree is at most $\frac{1}{\tau}$. As discussed above, at each level in the tree, the divide-and-conquer approach introduces a constant multiplicative approximation factor. These approximation factors accumulate, so the overall clustering cost for our algorithm's clustering of B_{m^*} is $2^{O(1/\tau)}$ times the cost of the optimal k-median clustering of B_{m^*}.

Now consider the non-expired points from bucket B_m. Invariant 3 guarantees that $f(B_m) \leq 2f(B_{m^*})$. This means that $f(B_m)$, our algorithm's cost for clustering all of B_m, is at most twice $f(B_{m^*})$, which is within a $2^{O(1/\tau)}$ factor of the optimal clustering cost for B_{m^*}. Only the cost of clustering the non-expired portion of B_m counts against us, but this can only be less than the clustering cost when summing over B_m in its entirety.

Therefore the costs of our algorithm's solution for the points in B_{m^*} and also for the points in B_m are both within a $2^{O(1/\tau)}$ factor of the optimal clustering cost for B_{m^*}. B_{m^*} is a subset of the active points, so the cost of the optimal k–medians clustering of all active points can only be greater than the cost of the optimal clustering of B_{m^*}. ■

Lemma 5 *The algorithm which maintains Invariant 4 has at most $O(\frac{1}{\tau}\log N)$ buckets at any time.*[2]

Proof Sketch: Invariant 4 guarantees that the number of buckets is at most $2\log R$ where R is our cost function over the entire sliding window. Lemma 4 proves that our cost function is at most $2^{O(1/\tau)}$ times the cost of the optimal k–medians solution. Since the optimal k–medians cost for N points is *poly*(N) this gives us that $\log R = O(\frac{1}{\tau}\log N)$. ■

3.3 Optimizing the Bicriteria k-Medians Algorithm

The above k–medians algorithm is efficient in terms of memory, but not in running time. After each element arrives, the algorithm checks all the buckets to see whether Invariant 4 is violated, in which case it combines two adjacent buckets to restore the invariant. The computation cost is dominated by the time taken to compute the $f(B_{i^*})$ values, the estimated costs of clustering the suffix buckets. The statistics for the suffix buckets can be generated incrementally while sweeping over the buckets.

[2]We assume that $d(x, y)$ is bounded by a polynomial in N.

Computing $f(B_{i^*})$ for a single bucket B_i involves clustering as many as N^τ medians per level at $\frac{1}{\tau}$ levels. Since this computation must be done once per bucket for $O(\frac{1}{\tau} logN)$ buckets, the total running time for each sweep over the buckets is $O(\frac{1}{\tau^3}N^{2\tau}\log N)$, assuming that a $O(n^2)$ algorithm is used as the clustering subroutine.

The running time is substantial given that a sweep over the buckets is performed each time a new data point arrives. We now optimize the algorithm to reduce the cost of maintaining the EH data structure to $\tilde{O}(k)$ amortized time per data point. To this end, we draw a distinction between data structure maintenance and output production. Our algorithm's goal is to be able to generate a near-optimal k–medians clustering of the most recent N data elements at any time. For efficiency, we assume that the algorithm will not actually be called upon to continually maintain a clustering of the sliding window at every time; rather, the algorithm should continually maintain sufficient statistics so as to be able, upon request, to quickly generate a valid clustering. Requests for the current clusters may come at arbitrary points in time, but we assume that they will not come too frequently.

We optimize the algorithm by processing new data points in batches. As these points arrive, they are simply added to bucket B_1 until it contains $\frac{k}{\tau^3}N^{2\tau}\log N$ points. No effort is made to maintain the invariants until the bucket "fills up," at which time the points in B_1 are clustered and replaced by k level-1 medians. However, the original data points are not yet discarded; they are retained until their bucket satisfies Invariant 3. After clustering the points in B_1, the other steps in the algorithm (discarding expired buckets and maintaining Invariant 4) are performed. Finally, a new empty bucket B_1 is created and the subscripts of the existing buckets are incremented.

Consider what happens to the invariants as points accumulate in bucket B_1. Since the value of $f(B_{i^*})$ is increasing for all $i > 1$, it is possible that Invariant 4 may no longer be satisfied. However, the temporary failure of this invariant is not important; recall that Invariant 4 is not essential for the correctness of the answer provided by the algorithm, but only to ensure that the number of buckets remains small. However, while the algorithm is filling up bucket B_1, the number of buckets does not increase, and as soon as the bucket is complete the invariant is restored by combining buckets as necessary. Thus, the number of buckets maintained by the algorithm is always $O(\frac{1}{\tau}\log N)$ and Lemma 5 is still valid although Invariant 4 may be temporarily violated.

Invariant 3 will not cause trouble for any bucket B_i $(i > 1)$ as bucket B_1 fills, because the increase in $f(B_{i^*})$ for these buckets only strengthens the invariant. However, this invariant is almost certain to be violated for bucket B_1. Since $f(B_{1^*}) = 0$, Invariant 3 fails to hold as soon as the clustering cost for bucket B_1 becomes non-zero, i.e., as soon as there are $k + 1$ distinct points in bucket B_1. Recall from the proof of Lemma 4 that the use that we made of Invariant 3 in the analysis was to ensure that the bucket cost $f(B_m)$ of the oldest bucket was not much more than the combined bucket cost $f(B_{m^*})$ of the other buckets. This was essential because we could bound the performance of the optimal algorithm in terms of $f(B_{m^*})$, but not in terms of $f(B_m)$ because $f(B_m)$ potentially includes the cost of clustering expired points. If the expired points are outliers, then $f(B_m)$ could be much larger than the cost of the optimal algorithm's solution over all non-expired data points. However, as long as we maintain the original data points from each bucket B_i that violates Invariant 3, this is a non-issue. If the invariant holds for the oldest bucket B_m, then we use the estimation procedure described earlier; if the invariant fails for B_m, then given the original data points for the bucket, we can distinguish the expired elements from the active ones and cluster only the latter.

We cluster the level-0 medians using the randomized algorithm from Indyk [19] with the local search algorithm from Charikar and Guha [6] as a subroutine. This procedure requires linear space and takes time $\tilde{O}(nk)$ (where n is the number of points that are clustered) while providing a constant factor approximation with high probability. All higher level (level-1 and above) medians are clustered using the $O(n^2)$ local search algorithm from Charikar and Guha [6]. While this algorithm uses up to $2k$ centers, the number can be reduced to k for the final answer via the primal-dual algorithm of Jain and Vazirani [20].

Putting everything together we have the following theorem. The memory bound follows from Lemma 5. The space used to maintain the original data points for buckets violating Invariant 3 dominates the space used to maintain the EH itself. The approximation guarantee follows from Lemma 4, while the maintenance time bound follows from the preceding discussion.

Theorem 4 *The algorithm presented above (including the batch-processing optimization) provides a $(2, 2^{O(1/\tau)})$ ($\tau < 1/2$) bicriteria approximation to the* SWKM *problem. It uses $O(k\frac{1}{\tau^4}N^{2\tau}\log^2 N)$ memory and requires amortized $\tilde{O}(k)$ maintenance time per data element.*

Finally, a note on the query time of our data structure. Whenever a current clustering is desired, we cluster the medians in the last bucket B_m, and then the active medians in the suffix buckets B_{m^*}. If bucket B_m violates Invariant 3 then we cluster only the active elements in it using the randomized algorithm from [19] to get their k-medians, which are then clustered with the rest of the active medians in the suffix bucket B_{m^*}. The total number of such medians is at most $O(\frac{1}{\tau^2}N^\tau \log N)$. The running time for clustering the active elements in B_m dominates, and the total time is $\tilde{O}(\frac{1}{\tau^4}N^{2\tau})$.

3.4 Approximate Clustering with Exactly k Clusters

We now present a new algorithm which builds upon the previous one and reduces the number of medians returned from $2k$ to k, while preserving the approximation guarantee of $2^{O(1/\tau)}$, but increasing the memory requirement by a factor $k^{\frac{1}{\tau}} \log N$.

The new algorithm is identical to the previous one, except for the following two things: (1) with every median we maintain additional information that lets us estimate to within a constant factor the number of active data points that are assigned to it, and (2) we change the estimation procedure that produces the solution whenever a request is made for the current clusters. Rather than separately clustering the last bucket B_m and the suffix bucket B_{m^*}, we cluster the medians from all the buckets together. However, the weights of the medians in bucket B_m are adjusted so that they reflect only the contribution of active data points, discounting the contribution of the expired points.

The algorithm maintains that both the cost of B_m and B_{m^*} are within $2^{O(1/\tau)}$ times the cost of the optimal k–medians solution for the current window. Using this fact and Theorem 3 it is easy to prove the following Lemma for the modified estimation procedure above.

Lemma 6 *The estimation procedure above produces a solution with k medians whose cost is $2^{O(1/\tau)}$ times the cost of the optimal k–medians solution for the current window.*

We now show how to estimate the number of active data points assigned to each median using at most $k^{1/\tau}\log N$ space per median. For any median c_j, consider the following imaginary stream $Z^{(j)}$ of $0-1$ values: For each actual data point x_i, $Z^{(j)}_i = 1$ if x_i was assigned to median c_j, and $Z^{(j)}_i = 0$ otherwise. Note that $Z^{(j)}_i$ will always be zero outside the interval covered by the bucket containing c_j. We would like to count the number of ones from the last N elements of $Z^{(j)}$. If the stream $Z^{(j)}$ were presented explicitly, this problem could be solved using an EH data structure that uses $O(\frac{1}{\epsilon}\log N)$ memory and provides an estimate at all times that has error at most ϵ, as shown in [9]. However, medians at level 2 and above are produced by the combination of lower-level medians. All points previously assigned to lower-level medians are now assigned to a higher-level median, but there is no way to reconstruct the exact arrival order of those points to simulate the stream $Z^{(j)}$ since the original points have been discarded. However, the EH for a median at level-l can be constructed from the EHs for the medians at level-$l-1$ that are assigned to it. The number of buckets of an EH for a median at level-l is given by $k^{l-1}\log N$. This gives the following theorem proved in the Appendix:

Theorem 5 *For $\tau < 1/2$, the SWKM algorithm provides a $2^{O(1/\tau)}$-approximation. It uses $O(\frac{k^{1/\tau}}{\tau^4}N^{2\tau}\log^2 N)$ memory and uses $\tilde{O}(k)$ amortized maintenance time per data element.*

References

[1] N. Alon, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments." In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, 1996, pages 20–29.

[2] S. Arora, P. Raghavan, and S. Rao. "Approximation Schemes for Euclidean k–Medians and Related Problems." In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)* 1998, pages 106–113.

[3] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. "Local Search Heuristics for k–Median and Facility Location Problems." In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 2001, pages 21–29.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. "Models and Issues in Data Stream Systems." To appear in *Proceedings of ACM SIGMOD Symposium on Principles of Databases Systems*, 2002. Available at `http://www-db.stanford.edu/stream/`.

[5] B. Babcock, M. Datar, and R. Motwani. "Sampling from a Moving Window over Streaming Data." In *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 633–634.

[6] M. Charikar and S. Guha. "Improved Combinatorial Algorithms for the Facility Location and k–Median Problems." In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science(FOCS)*, 1999, pages 378–388.

[7] M. Charikar, S. Guha, E. Tardos, and D. Shmoys. "A Constant-Factor Approximation Algorithm for the k–Median Problem (Extended Abstract)." In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, 1999, pages 1–10.

[8] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan, "Algorithms for Facility Location Problems with Outliers." In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001, pages 642–651.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining Stream Statistics over Sliding Windows." In *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 635–644.

[10] P. Domingos and G. Hulten. "Mining High-speed Data Streams." In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000, pages 71–80.

[11] P. Domingos, G. Hulten, and L. Spencer. "Mining Time-changing Data Streams." In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2001, pages 97–106.

[12] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. "An Approximate L1-Difference Algorithm for Massive Data Streams." In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999, pages 501–511.

[13] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. "Fast, Small-Space Algorithms for Approximate Histogram Maintenance." To appear in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

[14] S. Guha and N. Koudas. “Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation.” In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[15] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. “Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries.” In *Proceedings of the Annual Conference on Very Large Data Bases (VLDB)*, 2001, pages 79–88.

[16] S. Guha, N. Koudas, and K. Shim. “Data-Streams and Histograms.” In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 2001, pages 471–475.

[17] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan. “Clustering Data Streams.” In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pages 359–366.

[18] P. Indyk. “Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation.” In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pages 189–197.

[19] P. Indyk. “Sublinear Time Algorithms for Metric Space Problems.” In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, 1999, pages 428–434.

[20] K. Jain and V. Vazirani. “Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems.” In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999, pages 1–10.

[21] N. Mishra, D. Oblinger, and L. Pitt. “Sublinear Time Approximate Clustering.” In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001, pages 439–447.

[22] R. R. Mettu and C. G. Plaxton. “The Online k–Median Problem.” In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pages 339-348.

[23] L. O’Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. “Streaming-Data Algorithms for High-Quality Clustering.” In *Proceedings of the 18th Annual IEEE International Conference on Data Engineering (ICDE)*, 2001.

[24] M. Thorup. “Quick k–Median, k–Center, and Facility Location for Sparse Graphs.” In *Proceedings of the 28th Annual International Colloquium on Algorithms, Languages, and Programming (ICALP), LNCS*, 2001, pages 249–260.

[25] Stanford Stream Data Manager Project. `http://www-db.stanford.edu/stream`.

Appendix

A Proof of Lemma 1

Proof Sketch: First, note that $n_{i,j}$ and $\mu_{i,j}$ are correctly computed by the definitions of count and average. Define $\delta_i = \mu_i - \mu_{i,j}$ and $\delta_j = \mu_j - \mu_{i,j}$.

$$\begin{aligned}
V_{i,j} &= \sum_{x_l \in B_{i,j}} (x_l - \mu_{i,j})^2 \\
&= \sum_{x_l \in B_i} (x_l - \mu_i + \delta_i)^2 + \sum_{x_l \in B_j} (x_l - \mu_j + \delta_j)^2) \\
&= \sum_{x_l \in B_i} (x_l - \mu_i)^2 + 2\delta_i (x_l - \mu_i) + {\delta_i}^2 + \sum_{x_l \in B_j} (x_l - \mu_j)^2 + 2\delta_j (x_l - \mu_j) + {\delta_j}^2 \\
&= V_i + V_j + {\delta_i}^2 n_i + {\delta_j}^2 n_j + 2\delta_i \left(\sum_{x_l \in B_i} x_i - \sum_{x_l \in B_i} \mu_i \right) + 2\delta_j \left(\sum_{x_l \in B_j} x_j - \sum_{x_l \in B_j} \mu_j \right) \\
&= V_i + V_j + (\delta_i)^2 n_i + (\delta_j)^2 n_j \\
&= V_i + V_j + n_i \left(\frac{n_j(\mu_j - \mu_i)}{n_i + n_j}\right)^2 + n_j \left(\frac{n_i(\mu_i - \mu_j)}{n_i + n_j}\right)^2 \\
&= V_i + V_j + \frac{n_i n_j}{n_{i,j}} (\mu_i - \mu_j)^2
\end{aligned}$$

■

B Proof of Lemma 3

Proof Sketch: The proof follows from the following calculations.

$$\begin{aligned}
\sum_{x \in B} (x - \mu)^2 + |B|(a - \mu)^2 &= \sum_{x \in B} (x - \mu)^2 + (a - \mu)^2 \\
&= \sum_{x \in B} x^2 + 2\mu^2 + a^2 - 2x\mu - 2a\mu \\
&= \sum_{x \in B} x^2 + 2\mu(\mu - x) - 2a\mu + a^2 \\
&= \sum_{x \in B} x^2 - 2ax + a^2 \\
&= \sum_{x \in B} (x - a)^2
\end{aligned}$$

■

C Proof of Theorem 1

Proof Sketch: The memory usage is demonstrated by Lemma 2. Define $\delta = \mu_m - \mu_{\tilde{m}} = \mu_{\tilde{m}}^{EST} - \mu_{\tilde{m}}$. By the combination rule and our estimates for $B_{\tilde{m}}$,

$$\begin{aligned}
\mathrm{VAR}(t) - \widehat{\mathrm{VAR}}(t) &= \left(V_{\tilde{m}}^{EST} + V_{m^*} + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} (\mu_{\tilde{m}}^{EST} - \mu_{m^*})^2\right) - \left(V_{\tilde{m}} + V_{m^*} + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} (\mu_{\tilde{m}} - \mu_{m^*})^2\right) \\
&= (V_m/2 - V_{\tilde{m}}) + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} (2\delta(\mu_{\tilde{m}} - \mu_{m^*}) + \delta^2) \\
&= (V_m/2 - V_{\tilde{m}}) + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} \delta^2 + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} (2\delta(\mu_{\tilde{m}} - \mu_{m^*}))
\end{aligned}$$

We will show that each of the three additive terms in the error is small. Since $B_{\tilde{m}}$ is a subinterval of B_m we know that $V_{\tilde{m}} \leq V_m$. As variance is always non-negative, it follows that $|\frac{V_m}{2} - V_{\tilde{m}}| \leq \frac{V_m}{2}$. By Lemma 3 we know that $n_{\tilde{m}}(\mu_{\tilde{m}} - \mu_{\tilde{m}}^{EST})^2 = n_{\tilde{m}}(\mu_{\tilde{m}} - \mu_m)^2 \leq \sum_{x \in B_{\tilde{m}}}(x - \mu_m)^2 \leq V_m$, which implies that $|\frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} \delta^2| \leq V_m$. Define $c_2 = 3$, a constant derived from the analysis. For the third error term, consider two cases:

Case 1 ($|\mu_{\tilde{m}} - \mu_{m^*}| \leq |\frac{c_2 \delta}{\epsilon}|$): Then $|\frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} 2\delta(\mu_{\tilde{m}} - \mu_{m^*})| \leq \frac{n_{\tilde{m}} n_{m^*} 2c_2\delta^2}{(n_{\tilde{m}} + n_{m^*})\epsilon} \leq \frac{2c_2 V_m}{\epsilon}$.

Case 2 ($|\mu_{\tilde{m}} - \mu_{m^*}| > |\frac{c_2 \delta}{\epsilon}|$): The actual variance $\hat{\text{VAR}}(t)$ is at least $\frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}}(\mu_{\tilde{m}} - \mu_{m^*})^2$. The ratio between the third error term and the overall variance is at most $\frac{|2\delta(\mu_{\tilde{m}} - \mu_{m^*})|}{(\mu_{\tilde{m}} - \mu_{m^*})^2} \leq |\frac{2\delta}{\mu_{\tilde{m}} - \mu_{m^*}}| \leq \frac{2\epsilon}{c_2}$.

By Invariant 1 we know that $V_m \leq \frac{\epsilon^2}{9} V_{m^*}$. The first two error terms contribute a combined additive error of at most $\frac{3}{2}V_m$, which is a multiplicative error of at most $\frac{1}{6}\epsilon^2$ since $V_{m^*} \leq \hat{\text{VAR}}(t)$. The third error term contributes an additive error of at most $\frac{2c_2 V_m}{\epsilon}$ (in case 1), which represents a multiplicative error of at most $\frac{2}{3}\epsilon$. In case 2 the multiplicative error from the third error term is at most $\frac{2}{3}\epsilon$. We assume that $\epsilon \leq 1$ since otherwise the trivial algorithm that always returns zero suffices. In both cases we have the total error strictly less than an ϵ fraction of the overall variance, proving the theorem. ■

D Proof of Theorem 5

Proof Sketch: Lemma 6 proves the approximation guarantee of the modified algorithm. The memory usage of the modified algorithm, in terms of the number of medians that need to stored, is no different from the previous algorithm that provides a bicriteria approximation and whose performance is summarized in Theorem 4. What remains to prove is that the memory requirement of the EHs that are maintained with every level-1 and higher level median is no more than $k^{\frac{1}{\tau}} \log N$. To this end we make the following two observations about EHs over $0-1$ data streams:

1. Consider two Exponential Histograms (EH1 and EH2) corresponding to two $0-1$ data streams for the same error parameter ϵ and same window size N. If all the 1's in EH1 have arrival times strictly greater than the arrival times for 1's in EH2 then we can combine the two EH's to get an EH for the union of the two streams in time $O(\frac{1}{\epsilon}\log N)$. We term the original EHs as non-overlapping or disjoint. This "combination" can be easily achieved by placing the buckets of the two EHs one after the other and then making a sweep from right to left to combine the buckets, similar to all EHs, so that there are no more than $O(\frac{1}{\epsilon}\log N)$ buckets.

2. If the two data streams (respectively two EHs) are overlapping, i.e., they do not satisfy the property that all the 1's in one of them arrive before the 1's in other, then we can maintain two separate EHs corresponding to them to answer count queries over the union of the two streams.

When a higher level median is formed by clustering lower level medians we assign all the EHs of the lower level medians to the higher level medians. We combine as many EHs as possible which are non-overlapping. However, those that are overlapping are maintained separately. Thus, what we refer to as an EH of a median, in the discussion preceding the Theorem, is actually a collection of EHs, where the size of each EH is $O(\frac{1}{\epsilon}\log N)$. We now prove using induction, that we can combine EHs during clustering such that a median at level l needs to maintain no more than k^{l-1} EHs. Since $\frac{1}{\tau}$ is the maximum level of any median we get our final result.

Original points (level-0 medians) do not maintain an EH, they simply maintain their timestamps. The $Z^{(j)}$'s corresponding to them have 1 in exactly one position and zero everywhere else. When level-0 medians are clustered to form level-1 medians, for each level-1 median thus formed we can insert the level-0 medians assigned to it in the sorted (decreasing) order of timestamps so that we get a single EH for a level-1 median. Note, the level-1 medians obtained in this clustering may have overlapping EHs. Consider what happens when we cluster level-1 medians to form a level-2 median. Any level-2 median can be assigned at most k (O(k)) level-1 medians with overlapping EHs. We can form groups of overlapping EHs that all belong to the same time interval and arbitrarily index them from 1 to at most k, since there will at most k in a group. Now, all EHs with same the index across groups are non overlapping and we can combine them to form a single EH. Thus a level-2 median will have at most k EHs. In general, consider the case when level-l medians are clustered to form level-$l+1$ medians. By induction hypothesis each level-l median has at most k^{l-1} EHs which may be overlapping. Any level-$l+1$ median can be assigned at most k (O(k)) level-l medians which belong to same time interval and have overlapping EHs. Every level-l median will contribute at most k^{l-1} EHs. Thus if we form groups of overlapping EHs (that belong to same time interval) there will at most k^l in each group. We can index them from 1 to at most k^l. Again, all EHs with the same index are non-overlapping and can be combined into a single EH. Thus we get that a level-$l+1$ median will have at most k^l EHs.

A few final observations: For all the EHs we can set the error parameter ϵ to $\frac{1}{2}$. The combination procedure described above will be executed during the clustering of higher level (level-1 and above) medians. It is done in the last phase when points are assigned to the cluster centers. The time taken to assign every point is now $O(k^{\frac{1}{\tau}} \log N)$ instead of $O(1)$. Since we use a quadratic running time algorithm to cluster the higher level medians this does not affect the asymptotic running time for clustering provided $k^{\frac{1}{\tau}} \log N < N^{\tau}$. This completes the proof. ■

E Figures

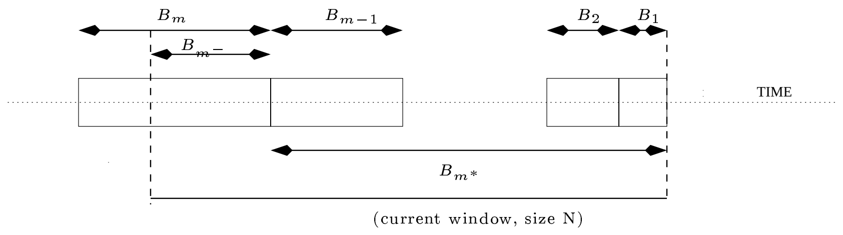

Figure 1: An illustration of the histogram.

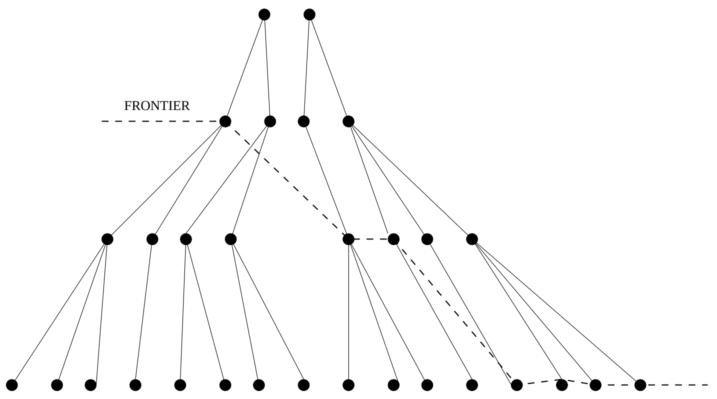

Figure 2: Contents of main memory : A frontier of the hierarchical clustering tree