Time-Decayed Correlated Aggregates over Data Streams *†

Graham Cormode
AT&T Labs–Research
graham@research.att.com

Srikanta Tirthapura   Bojian Xu
ECE Dept., Iowa State University
{snt,bojianxu}@iastate.edu

Abstract

Data stream analysis frequently relies on identifying correlations and posing conditional queries on the data after it has been seen. *Correlated aggregates* form an important example of such queries, which ask for an aggregation over one dimension of stream elements which satisfy a predicate on another dimension. Since recent events are typically more important than older ones, *time decay* should also be applied to downweight less significant values. We present space-efficient algorithms as well as space lower bounds for the time-decayed correlated sum, a problem at the heart of many related aggregations. By considering different fundamental classes of decay functions, we separate cases where efficient approximations with relative error or additive error guarantees are possible, from other cases where linear space is necessary to approximate. In particular, we show that no efficient algorithms with relative error guarantees are possible for the popular sliding window and exponential decay models, resolving an open problem. This negative result for the exponential decay holds even if the stream is allowed to be processed in multiple passes. The results are surprising, since efficient approximations are known for other data stream problems under these decay models. This is a step towards better understanding which sophisticated queries can be answered on massive streams using limited memory and computation.

Keywords: data stream, time decay, correlated, aggregate, sum

1 Introduction

Many applications such as Internet monitoring, information systems auditing, and phone call quality analysis involve monitoring massive data streams in real time. These streams arrive at high rates, and are too large to be stored in secondary storage, let alone in main memory. An example stream of VoIP call data records (CDRs) may have the call start time, end time, and packet loss rate, along with identifiers such as source and destination phone numbers. This stream can consist of billions of items per day. The challenge is to collect sufficient summary information about these streams in a single pass to allow subsequent post hoc analysis.

*A preliminary version of this paper is to appear in Proceedings of the 9th SIAM International Conference on Data Mining (SDM) 2009.

†The work of Tirthapura and Xu was supported in part by the National Science Foundation through grants 0520102, 0834743, 0831903.

There has been much research on estimating aggregates along a single dimension of a stream, such as the median, frequency moments, entropy, etc. However, most streams consist of multi-dimensional data. It is imperative to compute more complex multi-dimensional aggregates, especially those that can "slice and dice" the data across some dimensions before performing an aggregation, possibly along a different dimension. In this paper, we consider such *correlated aggregates*, which are a powerful class of queries for manipulating multi-dimensional data. These were motivated in the traditional OLAP model [3], and subsequently for streaming data [1, 8]. For example, consider the query on a VoIP CDR stream: "what is the average packet loss rate for calls within the last 24 hours that were less than 1 minute long"? This query involves a selection along the dimensions of call duration and call start time, and aggregation along the third dimension of packet loss rate. Queries of this form are useful in identifying the extent to which low call quality (high packet loss) causes customers to hang up. Another example is: "what is the average packet loss rate for calls started within the last 24 hours with duration greater than the median call length (within the last 24 hours)?", which gives a statistic to monitor overall quality for "long" calls. Such queries cannot be answered by existing streaming systems with guaranteed accuracy, unless they explicitly store all data for the last 24 hours, which is typically infeasible.

In this work, we present algorithms and lower bounds for approximating time-decayed correlated aggregates on a data stream. These queries can be captured by three main aspects: selection along one dimension (say x-dimension) and aggregation along a second dimension (say y-dimension) using time-decayed weights defined via a third (time) dimension. The time-decay arises from the fact that in most streams, recent data is naturally more important than older data, and in computing an aggregate, we should give a greater weight to more recent data. In the examples above, the time decay arises in the form of a sliding window of a certain duration (24 hours) over the data stream. More generally, we consider arbitrary time-decay functions which return a weight for each element as a non-increasing function of its age—the time elapsed since the element was generated. Importantly, the nature of the time-decay function will determine the extent to which the aggregate can be approximated.

We focus on the *time-decayed correlated sum* (henceforth referred to as DCS), which is a fundamental aggregate, interesting in its own right, and to which other aggregates can be reduced. An exact computation of the correlated sum requires multiple passes through the stream, even with no time-decay where all elements are weighted equally. Since we can afford only a single pass over the stream, we will aim for approximate answers with accuracy guarantees. In this paper, we present the first streaming algorithms for estimating the DCS of a stream using limited memory, with such guarantees. Prior work on correlated aggregates either did not have accuracy guarantees on the results [8] or else did not allow time-decay [1]. We first define the stream model and the problem more precisely, and then present our results.

1.1 Problem Formulation

We consider a stream $R = e_1, e_2, \ldots, e_n$. Each element e_i is a (v_i, w_i, t_i) tuple, where $v_i \in [m]$ is a value or key from an ordered domain; positive integer w_i is the initial weight; and t_i is the timestamp at which the element was created or observed, also assumed to be a positive integer. For example, in a stream of VoIP call records, there is one stream element per call, where t_i is the time the call was placed, v_i is the duration of the call, and w_i the packet loss rate.

In the synchronous model, the arrivals are in timestamp order: $t_i \leq t_{i+1}$, for all i. In the strictly synchronous version, there is exactly one arrival at each time step, so that $t_i = i$. We also consider the general asynchronous streams model [12, 2, 5], where the order of receipt of stream elements is not necessarily the same as the increasing order of their timestamps. Therefore, it is possible that $t_i > t_j$ while $i < j$. Such asynchrony is inevitable in many applications: in the VoIP call example, CDRs maybe collected at different locations in the network, and in sending this data to a central server, the interleaving of many (possibly synchronous) streams could result in an asynchronous stream.

The aggregates will be time-decayed, i.e. elements with earlier timestamps will be weighted lower than elements with more recent timestamps. The exact model of decay is specified by the user through a *time-decay* function.

Definition 1.1. *A function $f(x)$, $x \geq 0$, is called a* time-decay function *(or just a* decay function*) if: (1) $0 \leq f(x) \leq 1$ for all $x > 0$; (2) if $x_1 \leq x_2$, then $f(x_1) \geq f(x_2)$.*

At time $t \geq t_i$ the *age* of $e_i = (v_i, w_i, t_i)$ is defined as $t - t_i$, and the *decayed weight* of e_i is $w_i \cdot f(t - t_i)$.

Time-Decayed Correlated Sum. The query for the time-decayed correlated sum over stream R under a prespecified decay-function f is posed at time t, provides a parameter $\tau \geq 0$, and asks for S^f_τ, defined as follows:

$$S^f_\tau = \sum_{e_i \in R | v_i \geq \tau} w_i \cdot f(t - t_i)$$

A correlated aggregate query could be: "What is the average packet loss rate for all calls which started in the last 24 hours, and were more than 30 minutes in length?". This query can be split into two sub-queries: The first sub-query finds the number of stream elements (v_i, w_i, t_i) which satisfy $v_i > 30$, and $t_i > t - 24$ where t is the current time in hours. The second sub-query finds the sum of w_is for all elements (v_i, w_i, t_i) such that $v_i > 30$ and $t_i > t - 24$. The average is the ratio of the two answers.

DCS lies at the heart of many other aggregates. Example time-decayed aggregates that can reduced to DCS are the following:

- The *time-decayed relative frequency* of a value v, which is given by $(S^f_v - S^f_{v+1})/S^f_0$.
- The *sum of time-decayed weights* of elements in the range $[l, r]$, which is $S^f_l - S^f_{r+1}$.
- The *time-decayed frequency* of range $[l, r]$, which is $(S^f_l - S^f_{r+1})/S^f_0$.
- The *time-decayed ϕ-heavy hitters*, which are all the v's such that the time decayed relative frequency of v is at least ϕ.
- The *time decayed correlated ϕ-quantile*, which is the largest v, such that $(S^f_0 - S^f_v)/S^f_0 \leq \phi$.

Time-Decayed Correlated Count. An important special case of DCS is the *time-decayed correlated count* (henceforth referred to as DCC), where all the weights w_i are assumed to be 1. The correlated count C_τ^f is therefore:

$$C_\tau^f = \sum_{e_i \in R | v_i \geq \tau} f(t - t_i)$$

1.2 Decay Functions Classes

We define classes of decay functions, which cover popular time decays from prior work.

Converging decay. A decay function $f(x)$ is a converging decay function if $f(x+1)/f(x)$ is non-decreasing with x. Intuitively, the relative weights of elements with different timestamps under a converging decay function get closer to each other as time goes by. As pointed out by Cohen and Strauss [4], this is an intuitive property of a time-decay function in several applications. Many popular decay functions, such as polynomial decay (functions $f(x) = (x+1)^{-a}$ based on an exponent $a > 0$) are converging decay functions. Converging decay also includes the no decay case: $f(x) \equiv 1$.

Exponential decay. Given a constant $\alpha > 0$, the exponential decay function is defined as $f(x) = 2^{-\alpha x}$. Exponential decay with a different base can also be written in this form, since $a^{-\lambda x} = 2^{-\lambda \log_2(a) x}$. As $f(x+1)/f(x)$ is a constant, exponential decay qualifies as a converging decay function.

Super-exponential decay. A decay function $f(x)$ is super-exponential, if there exist constants $\sigma > 1$ and $c \geq 0$, such that for every $x \geq c$, $f(x)/f(x+1) \geq \sigma$. Examples of such decay functions include: (1) *polyexponential decay* [4]: $f(x) = (x+1)^k 2^{-\alpha x}/k!$ where $k > 0$, and $\alpha > 0$ are constants. (2) $f(x) = 2^{-\alpha x^\beta}$, where $\alpha > 0$ and $\beta > 1$.

Finite decay. A decay function f is defined to be a *finite decay function* with age limit N, if there exists $N \geq 0$ such that for $x > N$, $f(x) = 0$, and for $x \leq N$, $f(x) > 0$. Examples of finite decay include (1) sliding window decay: $f(x) = 1$ for $x \leq N$ and 0 otherwise, where the age limit N is the window size. (2) Chordal decay [4]: $f(x) = 1 - x/N$ for $0 \leq x \leq N$ and 0 otherwise, with an age limit of $N-1$. Obviously, no finite decay function is a converging decay function, since $f(N+1)/f(N) = 0$ while $f(N)/f(N-1) > 0$.

1.3 Contributions

Our main result is that there exist small space algorithms for approximating DCS over an arbitrary decay function f with a small *additive* error. But, the space cost of approximating DCS with a small *relative* error depends strongly on the nature of the decay function—this is possible on some classes of functions using small space, while for other classes, including sliding window and exponential decay, this is provably impossible in sublinear space. More specifically, we show:

1. For *any* decay function f, there is a randomized algorithm for approximating DCS with bounded additive error guarantee which uses space logarithmic in the size of the stream. This significantly

improves on previous work [8], which presented heuristics only for sliding window decay. (Section 3.1)

2. On the other hand, for any *finite* decay function, we show that approximating DCS with a small *relative* error needs space linear in the size of the elements whose ages are not larger than the age limit of the decay function. Because sliding window decay is a finite decay function, the above two results resolve the open problem posed in [1], which was to determine the space complexity of approximating the correlated sum under sliding window decay. (Section 4.1)
3. For any non-exponential converging decay function, there is an algorithm for approximating DCS to within a small relative error using space logarithmic in the stream size, and logarithmic in the "rate" of the decay function. (Section 3.2)
4. For any exponential decay function and super-exponential decay function, we show that the space complexity of approximating DCS with a small relative error is linear in the stream size, in the worst case, even if multi-pass processing of the stream is allowed. This may be surprising, since there are simple and efficient solutions for maintaining exponentially decayed sum exactly in the non-correlated case. (Section 4.2)

We evaluate our techniques over real and synthetic data in Section 5, and observe that they can effectively summarize massive streams in tens of kilobytes.

2 Prior Work

Concepts of correlated aggregation in the (non-streaming) OLAP context appear in [3]. The first work to propose correlated aggregation for streams was Gehrke *et al.* [8]. They assumed that data was locally uniform to give heuristics for computing the non-decayed correlated sum where the threshold (τ) is either an extrema (min, max) or the mean of the all the received values (v_i's). For the sliding window setting, they simply partition the window into fixed-length intervals, and make similar uniformity assumptions for each interval. No strong guarantee on the answer quality are provided by any of these approaches. Subsequently, Ananthakrishna *et al.* [1] presented summaries that estimate the non-decayed correlated sum with *additive error* guarantees. The problem of tracking sliding window based correlated sums with quality guarantees was given as an open problem in [1]. We show that this relative error guarantees are not possible while using small space, whereas additive guarantees can be obtained.

Xu *et al.* [12] proposed the concept of asynchronous streams. They gave a randomized algorithm to approximate the sum and the median over sliding windows. Busch and Tirthapura [2] later gave a deterministic algorithm for the sum. Cormode *et al.* [6, 5] gave algorithms for general time decay based aggregates over asynchronous streams. By defining timestamps appropriately, *non-decayed* correlated sum can be reduced to the sum of elements within a sliding window over an asynchronous stream. As a result, relative error bounds follow from bounds in [12, 2, 6, 5]. But these methods do not extend to accurately estimating DCS or DCC.

Datar *et al.* [7] presented a bucket-based technique called *exponential histograms* for sliding windows on synchronous streams. This approximates counts and related aggregates, such as sum and ℓ_p norms. Gibbons and Tirthapura [9] improved the worst-case performance for counts using a data structure called a *wave*. Going beyond sliding windows, Cohen and Strauss [4] formalized time-decayed data aggregation, and provided strong motivating examples for non-sliding window decay. All these works emphasized the time decay issue, but did not consider the problems of correlated aggregate computation.

3 Upper Bounds

In this section, we present algorithms for approximating DCS over a stream R. The main results are: (1) For an arbitrary decay function f, there is a small space streaming algorithm to approximate S_τ^f with a small additive error. (2) For any non-exponential *converging* decay function f, there is small space streaming algorithm to approximate S_τ^f with a small relative error.

3.1 Additive Error

A predicate $P(v,w)$ is a binary function of v and w, used to select certain items. For example, (1) the predicate could select only those items with $v > 1000$ by returning 1 for those items, and 0 for others; (2) the predicate could select only those items with $w < 100$ similarly; and (3) the predicate could select only those items with both $v > 1000$ and $w < 100$, and so on. The time-decayed selectivity Q of a predicate $P(v,w)$ on a stream R of (v,w,t) tuples is defined as

$$Q = \frac{\sum_{(v,w,t)\in R} P(v,w)\cdot w \cdot f(c-t)}{\sum_{(v,w,t)\in R} w \cdot f(c-t)}$$

The decayed sum S is defined as:

$$S = \sum_{(v,w,t)\in R} w \cdot f(c-t)$$

Note that $S = S_0^f$. We use the following results on time-decayed selectivity estimation from [6] in our algorithm for approximating DCS with a small additive error.

Theorem 3.1 (Theorems 4.1, 4.2, 4.3 from [6])**.** *Given* $0 < \varepsilon < 1$ *and probability* $0 < \delta < 1$*, there exists a small space sketch of size* $O(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta} \cdot \log M)$ *that can be computed in one pass from stream R, where M is an upper bound on S. For any decay function f given at query time: (1) the sketch can return an estimate* $\widehat{S}$ *for S such that* $\Pr[|\widehat{S} - S| \leq \varepsilon S] \geq 1-\delta$. *(2) Given predicate* $P(v,w)$ *at query time, the sketch gives an estimate* $\widehat{Q}$ *for the decayed selectivity Q, such that* $\Pr[|\widehat{Q} - Q| \leq \varepsilon] \geq 1-\delta$.

The sketch designed in [6] can be thought of as computing a set of fixed-sized random samples of the stream. Each successive sample is chosen with decreasing probability, so for a unit-weight stream element, its probability of selection in each successive sample is $1, \frac{1}{2}, \frac{1}{4}, \ldots$. For non-unit weight elements, the probability of selecting a stream element into the sample is also proportional to the decayed weight of the element. The result stated in the above theorem allows DCS to be additively approximated:

Theorem 3.2. *For an arbitrary decay function f, there exists a small space sketch of R that can be computed in one pass over the stream. At any time instant, given a threshold τ, the sketch can return $\widehat{S}^f_\tau$, such that $|\widehat{S}^f_\tau - S^f_\tau| \le \varepsilon S^f_0$ with probability at least $1-\delta$. The space complexity of the sketch is $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\cdot\log M)$, where M is an upper bound on S^f_0.*

Proof. We run the sketch algorithm in [6] on stream R, with approximation error $\varepsilon/3$ and failure probability $\delta/2$. Let this sketch be denoted by $\mathscr{K}$. Where the function f is implicit, we can drop it from our notation, and simply write $\widehat{S}_\tau$, S_τ in place of $\widehat{S}^f_\tau$, S^f_τ respectively.

Given τ at query time, we define a predicate P for the selectivity estimation as: $P(v,w) = 1$, if $v \ge \tau$, and $P(v,w) = 0$ otherwise. The selectivity of P is $Q = S_\tau/S$. Then $\mathscr{K}$ can return estimates $\widehat{Q}$ of Q and $\widehat{S}$ of S such that

$$\Pr[|\widehat{Q} - Q| > \varepsilon/3] \le 1 - \delta/2 \tag{1}$$

$$\Pr[|\widehat{S} - S| > \varepsilon S/3] \le 1 - \delta/2 \tag{2}$$

Our estimate $\widehat{S}_\tau$ is given by $\widehat{S}_\tau = \widehat{S}\cdot\widehat{Q}$. From (1) and (2), and using the union bound on probabilities, we get that the following events are both true, with probability at least $1-\delta$.

$$Q - \varepsilon/3 \quad \le \widehat{Q} \quad \le Q + \varepsilon/3 \tag{3}$$

$$S(1-\varepsilon/3) \quad \le \widehat{S} \quad \le S(1+\varepsilon/3) \tag{4}$$

Using the above, and using $Q = S_\tau/S$, we get

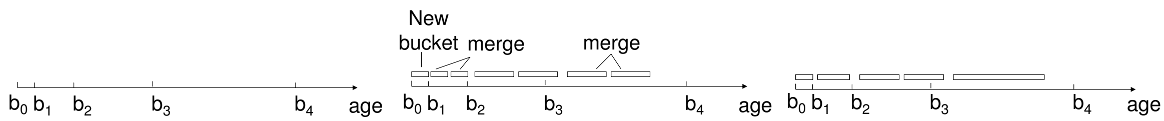
$$\widehat{S}_\tau \le \left(\frac{S_\tau}{S} + \varepsilon/3\right)\cdot S \cdot (1+\varepsilon/3) = S_\tau + \frac{S_\tau\varepsilon}{3} + S\left(\frac{\varepsilon}{3} + \frac{\varepsilon^2}{9}\right) \le S_\tau + \varepsilon S$$

In the last step of the above inequality, we have used the fact $S_\tau \le S$ and $\varepsilon < 1$. Similarly, we get that if (3) and (4) are true, then, $\widehat{S}_\tau \ge S_\tau - \varepsilon S$, thus completing the proof that $\mathscr{K}$ can (with high probability) provide an estimate $\widehat{S}^f_\tau$ such that $|\widehat{S}^f_\tau - S^f_\tau| \le \varepsilon S^f_0$ □

An important feature of this approach, made possible due to the flexibility of the sketch in Theorem 3.1, is that it allows the decay function f to be specified at query time, i.e. after the stream R has been seen. This allows for a variety of decay models to be applied in the analysis of the stream after the fact. Further, since the sketch is designed to handle asynchronous arrivals, the timestamps can be arbitrary and arrivals do not need to be in timestamp order.

3.2 Relative Error

In this section, we present a small space sketch that can be maintained over a stream R with the following properties. For an arbitrary *converging* decay function f which is known beforehand, and a parameter τ which is provided at query time, the sketch can return an estimate $\widehat{S}^f_\tau$ which is within a small relative error of S^f_τ. The space complexity of the sketch depends on f.



(a) Regions (b) Buckets & Regions (c) Buckets & Regions after Merge

Figure 1: Weight-based merging histograms.

The idea behind the sketch is to maintain multiple data structures each of which solves the undecayed correlated sum, and partition stream elements across different data structures, depending on their timestamps, following the approach of the Weight-Based Merging Histogram (WBMH), due to Cohen and Strauss [4]. In the rest of this section, we first give high level intuition, followed by a formal description of the sketch, and a correctness proof. Finally, we describe enhancements that allow faster insertion of stream elements into the sketch.

3.2.1 Intuition

We first describe the weight-based merging histogram [4]. The histogram partitions the stream elements into buckets based on their age. Given a decay function f, and parameter ε_1, the sequence $b_i, i \geq 0$ is defined as follows: $b_0 = 0$, and for $i > 0$, b_i is defined as the largest integer such that $f(b_i - 1) \geq \frac{f(b_{i-1})}{1+\varepsilon_1}$ (Figure 1(a)).

For simplicity, we first describe the algorithm for the case of a strictly synchronous stream, where the timestamp of a stream element is just its position in the stream. We later discuss the extension to asynchronous streams. Let G_i denote the interval $[b_i, b_{i+1})$ so that $|G_i| = b_{i+1} - b_i$. Once the decay function is given, the G_is are fixed and do not change over time. The elements of the stream are grouped into regions based on their age. For $i \geq 0$, region i contains all stream elements whose age lies in interval G_i.

For any i, we have $f(b_i) < \frac{f(b_0)}{(1+\varepsilon_1)^i}$, and thus we get $i < \log_{1+\varepsilon_1}\left(\frac{f(0)}{f(b_i)}\right)$. Since the age of an element cannot be more than n, $b_i \leq n$. Thus we get that the total number of regions is no more than $\beta = \lceil \log_{1+\varepsilon_1}\left(\frac{f(0)}{f(n)}\right) \rceil$. From the definition of the b_is, we also have the following fact.

Fact 3.1. *Suppose two stream elements have ages a_1 and a_2 so that a_1 and a_2 fall within the same region. Then,*

$$\frac{1}{1+\varepsilon_1} \leq \frac{f(a_1)}{f(a_2)} \leq 1 + \varepsilon_1$$

The data structure maintains a set of *buckets*. Each bucket groups together stream elements whose timestamps fall in a particular range, and maintains a small space summary of these elements. We say that the bucket is “responsible” for this range of timestamps (or equivalently, a range of ages).

Suppose that the goal was to maintain S_0^f, just the time-decayed sum of all stream elements. If the current time c is such that $c \mod b_1 = 0$, then a new bucket is created for handling future elements (Figure 1(b)). The algorithm ensures that the number of buckets does not grow too large through the following rule: if two adjacent buckets are such that the age ranges that they are responsible for are both contained within the same region, then the two buckets are merged into a single bucket. The count within the resulting bucket is equal to the sum of the counts of the two buckets, and the resulting bucket is responsible for the union of the ranges of timestamps the two buckets were responsible for (Figures 1(c)).

Due to the merging, there can be at most 2β buckets: one bucket completely contained within each region, and one bucket straddling each boundary between two regions. From Fact 3.1, the weights of all elements contained within a single bucket are close to each other, and since f is a converging decay function, this remains true as the ages of the elements increase. Consequently, WBMH can approximate S_0^f with ε_1 relative error by treating all elements in each bucket as if they shared the smallest timestamp in the range, and scaling the corresponding weight by the total count.

However, this does not solve the more general DCS problem, since it does not allow filtering out elements whose values are smaller than τ. We extend the above data structure to the DCS problem by embedding within each bucket a data structure that can answer the (undecayed) correlated sum of all elements that were inserted into this bucket. This data structure can be any of the algorithms that can estimate the sum of elements within a sliding window on asynchronous streams, including [12, 5, 2]: values of the elements are treated as timestamps, and a window size $m-\tau+1$ is supplied at query time (where m is an upper bound on the value).

These observations yield our new algorithm for approximating S_τ^f. We replace the simple count for each bucket in the WBMH with a small space sketch, from any of [12, 5]. We will not assume a particular sketch for maintaining the information within a bucket. Instead, our algorithm will work with any sketch that satisfies the following properties—we call such a sketch a "bucket sketch". Let ε_2 denote the accuracy parameter for such a bucket sketch.

1. The bucket sketch must concisely summarize a stream of (v,w) pairs using space polylogarithmic in the stream size. Given parameter $\tau \geq 0$ at query time, the sketch must return an estimate for $\sum_{v \geq \tau} w$, such that relative error of the estimate is within ε_2.
2. It must be possible to merge two bucket sketches easily into a single sketch. More precisely, suppose that S_1 is the sketch for a set of elements R_1 and S_2 is the sketch for a set of elements R_2, then it must be possible to merge together S_1 and S_2 to get a single sketch denoted by $S = S_1 \cup S_2$, such that S retains Property 1 for the set of elements $R_1 \cup R_2$.

The analysis of the sketch proposed in [12] explicitly shows that the above properties hold. Likewise, the sketch designed in [5] also has the necessary properties, since it is built on with multiple instances of q-digest summaries [11] which are themselves mergable. The different sketches have slightly different time and space complexities; we state and analyze our algorithm in terms of a generic bucket sketch, and subsequently describe the cost depending on the choice of sketch.

3.2.2 Formal Description and Correctness

Recall that ε is the required bound on the relative error. Our algorithm combines two data structures: the WBMH with accuracy parameter $\varepsilon_1 = \varepsilon/2$; and the bucket sketches with accuracy parameter $\varepsilon_2 = \varepsilon/2$. The initialization is shown in the SETBOUNDARIES procedure (Figure 2), which creates the regions G_i by

Algorithm 3.1: SETBOUNDARIES(ε)

comment: create $G_0, G_1, \ldots, G_\beta$ using $\varepsilon_1 = \varepsilon/2$
$b_0 \leftarrow 0$;
for $1 \le i \le \beta$
  do $b_i \leftarrow \max_x\{x|(1+\frac{\varepsilon}{2})f(x-1) \ge f(b_{i-1})\}$
comment: x are integers
$j \leftarrow -1$;
comment: j is the index of the active bucket for new elements

Figure 2: SETBOUNDARIES routine to initialize regions.

selecting $b_0, \ldots, b_\beta$. For simplicity of presentation, we have assumed that the maximum stream length n is known beforehand, but this is not necessary — the b_i's can be generated incrementally, i.e., b_i does not need to be generated until element ages exceeding b_{i-1} have been observed.

Figure 3 shows the PROCESSELEMENT procedure for handling a new stream element. Whenever the current time t satisfies $t \mod b_1 = 0$, we create a new bucket to summarize the elements with timestamps from t to $t+b_1-1$ and seal the last bucket which was created at time $t-b_1$. The procedure FINDREGIONS(t) returns the set of regions that contain buckets to be merged at time t. In the next section we present novel methods to implement this requirement efficiently. Figure 4 shows the procedure RETURNAPPROXIMATION which generates the answer for a query for S^f_τ at time t. Each bucket returns an estimate for the total undecayed weights of the elements that were inserted in the bucket and whose values are not smaller than τ. Each of these estimates is then scaled down by a factor of the decay function value using the corresponding $t-F_B$ as the parameter of the decay function. The summation of these scaled estimates are returned as the estimate for S^f_τ.

Theorem 3.3. *If f is a converging decay function, for any τ given at any time t, the algorithm specified in Figure 2, 3 and 4 can return $\widehat{S^f_\tau}$, such that $(1-\varepsilon)S^f_\tau \le \widehat{S^f_\tau} \le (1+\varepsilon)S^f_\tau$.*

Proof. For the special converging decay function where $f(x) \equiv 1$ (no decay), then WBMH has only one region and one bucket. So the algorithm reduces to a single bucket sketch. This sketch can directly provide an $\varepsilon_2 = \varepsilon/2$ relative error guarantee for the estimate of S^f_τ.

The broader case is where $f(x+1)/f(x)$ is non-decreasing with x. Let $\{B_1, \ldots, B_k\}$ be the set of sketch buckets at query time t. Let $R_i \subseteq R$ be the substream that was inserted into B_i, $1 \le i \le k$. Since every stream element is inserted into exactly one sketch bucket at any time, the R_is partition R: $\bigcup_{i=1}^k R_i = R$ and $R_i \cap R_j = \emptyset$ if $i \ne j$. Note that merging two buckets just creates a new bucket for the union of the two underlying substreams. Let $S^f_{\tau,i} = \sum_{e_j \in R_i | v_j \ge \tau} w_j f(t-j)$ be the DCS of R_i at time t, $1 \le i \le k$, so $S^f_\tau = \sum_{i=1}^k S^f_{\tau,i}$. We first consider the accuracy of the estimate for each $S^f_{\tau,i}$ using sketch bucket B_i, $1 \le i \le k$.

For each $(v_j, w_j, j) \in R_i$ at any query time t, since $F_{B_i} \le j \le L_{B_i}$ which implies $f(t-F_{B_i}) \le f(t-j) \le$

$f(t-L_{B_i})$, and $f(t-L_{B_i})/(1+\varepsilon_1) \le f(t-F_{B_i})$ (using Fact 3.1), we have

$$\frac{w_j f(t-j)}{1+\varepsilon_1} \le \frac{w_j f(t-L_{B_i})}{1+\varepsilon_1} \le w_j f(t-F_{B_i}) \le w_j f(t-j)$$

therefore,

$$\frac{1}{1+\varepsilon_1} \sum_{e_j \in R_i | v_j \ge \tau} w_j f(t-j) \le f(t-F_{B_i}) \sum_{e_j \in R_i | v_j \ge \tau} w_j \le \sum_{e_j \in R_i | v_j \ge \tau} w_j f(t-j)$$

i.e.,

$$\frac{1}{1+\varepsilon_1} S^f_{\tau,i} \le f(t-F_{B_i}) Q_i \le S^f_{\tau,i} \text{ , where } Q_i = \sum_{e_j \in R_i | v_j \ge \tau} w_j \tag{5}$$

Also since sketch bucket B_i can return $\widehat{Q}_i$ such that [12, 5]

$$(1-\varepsilon_2) Q_i \le \widehat{Q}_i \le (1+\varepsilon_2) Q_i \tag{6}$$

Combining 5 and 6, we have

$$\frac{1-\varepsilon_2}{1+\varepsilon_1} S^f_{\tau,i} \le \widehat{Q}_i \cdot f(t-F_{B_i}) \le (1+\varepsilon_2) S^f_{\tau,i}.$$

Now we sum all the $S^t_{\tau,i}$ together, $i = 1, 2, \ldots, k$, we get

$$\frac{1-\varepsilon_2}{1+\varepsilon_1} \sum_{i=1}^{k} S^f_{\tau,i} \le \sum_{i=1}^{k} \widehat{Q}_i \cdot f(t-F_{B_i}) \le (1+\varepsilon_2) \sum_{i=1}^{k} S^f_{\tau,i}.$$

i.e.,

$$\frac{1-\varepsilon_2}{1+\varepsilon_1} S^f_{\tau} \le \widehat{S^f_{\tau}} \le (1+\varepsilon_2) S^f_{\tau}$$

Using the facts $0 < \varepsilon < 1$ and $\varepsilon_1 = \varepsilon_2 = \varepsilon/2$, we get the stated accuracy guarantee. □

3.2.3 Fast Bucket Merging

At every clock tick the WBMH maintenance algorithm needs to check whether there are buckets that need to be merged. A naive solution is to go through all the buckets and merge those that are covered by the same region. This procedure can severely reduce the speed of stream processing. In this section we present an algorithm which, given the clock time t, can efficiently return the set of regions that have buckets to be merged at time t.

Definition 3.1 (Sketch bucket B's capacity $|B|$)**.** *The capacity of bucket B is given by $|B| = L_B - F_B + 1$, where L_B and F_B are the largest and smallest timestamps of the elements that were inserted into B (as in Figure 3).*

Definition 3.2 (Bucket capacity in the ith region)**.** *Define $I_0 = 1$. For $0 < i < \beta$, let $I_i = |B|$, where B is any bucket such that $t - F_B = b_i$ for some value of t.*

Algorithm 3.2: PROCESSELEMENT$((v_i, w_i, i))$

---

if $i \mod b_1 = 0$

then $\begin{cases} j \leftarrow j+1 \\ \text{Initialize a new bucket sketch } B_j \text{ with accuracy } \varepsilon/2 \\ F_{B_j} \leftarrow i \\ L_{B_j} \leftarrow i+b_1-1 \\ \textbf{comment: } \text{Set timestamp range covered by } B_j \end{cases}$

Insert (v_i, w_i) into B_j

for each $g \in$ FINDREGIONS(i)

comment: Set of regions with buckets to be merged at time i

do $\begin{cases} b_{min} \leftarrow \min_t\{t | t \in G_g\} \\ b_{max} \leftarrow \max_t\{t | t \in G_g\} \\ \textbf{comment: } \text{Find the left and right timestamp boundary of region } G_g \\ \text{Find buckets } B' \text{ and } B'', \text{ such that} \\ b_{min} \leq (i - L_{B'}) < (i - F_{B'}) < (i - L_{B''}) < (i - F_{B''}) \leq b_{max} \\ \textbf{comment: } \text{Find buckets covered by } G_g \\ B \leftarrow B' \cup B'' \\ \textbf{comment: } \text{merge two buckets} \\ F_B \leftarrow F_{B''} \\ L_B \leftarrow L_{B'} \\ \text{Drop } B' \text{ and } B'' \end{cases}$

Figure 3: PROCESSELEMENT routine to handle updates

In the next lemma, we show each I_i is a constant, $0 < i < \beta$, and can be directly computed.

Lemma 3.1. *For $0 < i < \beta$, $I_i = \lfloor |G_{i-1}|/I_{i-1} \rfloor \cdot I_{i-1}$*

Proof. The lemma is proved by induction. For the base case, since the capacity of the new bucket created in G_0 is exactly equal to $|G_0|$, no merges can happen in G_0, so $I_1 = |G_0| = \lfloor |G_0|/I_0 \rfloor \cdot I_0$. For the inductive step, suppose the claim is true for some i, i.e.,, I_i is a constant, which implies at most $\lfloor |G_i|/I_i \rfloor$ buckets of capacity I_i can be merged within G_i. The new bucket from merging therefore has capacity $\lfloor |G_i|/I_i \rfloor I_i = I_{i+1}$, which is a constant. This completes the proof. □

In the next lemma, we show that given I_i we can directly find the sequence of time points at which G_i has buckets to be merged.

Lemma 3.2. *For each $i \in \{j | j = 0$ or $\lfloor |G_j|/I_i \rfloor < 2\}$, G_i has no buckets to be merged at any time; for each $i \in \{j | j > 0$ and $\lfloor |G_j|/I_i \rfloor \geq 2\}$, G_i has buckets to be merged at time $\{b_i + (k\lfloor |G_i|/I_i \rfloor + j)I_i\}$, for each j and*

Algorithm 3.3: ReturnApproximation(τ,t)

Let the set of buckets be: $\{B_1, B_2, \ldots, B_k\}$
comment: for some k, $1 \leq k \leq 2\beta$;
$s \leftarrow 0$;
for $1 \leq i \leq k$
do $\begin{cases} \text{Let } \widehat{Q}_i \text{ be result for } B_i \text{ using } m-\tau+1 \text{ as window size} \\ s \leftarrow s + \widehat{Q}_i \cdot f(t - F_{B_i}); \end{cases}$
comment: Approx sum of element weights in B_i with $v_i \geq \tau$
return $(\widehat{S}^f_\tau = s)$;

Figure 4: ReturnApproximation routine to estimate S^f_τ

k, $2 \leq j \leq \lfloor |G_i|/I_i \rfloor$, $k \geq 0$.

Proof. The new bucket created in G_0 has capacity equal to $|G_0|$, so G_0 does not have any buckets to be merged at any time. For $i > 0$, if $\lfloor |G_i|/I_i \rfloor < 2$, then G_i will not have the chance to have two buckets of capacity I_i to be merged at any time. Now we consider the case where $\lfloor |G_i|/I_i \rfloor \geq 2$ and $i > 0$. G_i has its first whole bucket at time $t = b_i + I_i$. Note that within G_i at most $\lfloor |G_i|/I_i \rfloor$ buckets that enter G_i can be merged together. Thus, (1) at time $t = b_i + 2I_i, b_i + 3I_i, \ldots, b_i + \lfloor |G_i|/I_i \rfloor \cdot I_i$, buckets can be merged within G_i; (2) This sequence of merge operations repeats every $\lfloor |G_i|/I_i \rfloor \cdot I_i$ clock ticks, meaning G_i has buckets to be merged at times $\{b_i + (k\lfloor |G_i|/I_i \rfloor + j)I_i\}$, for each j and k, $2 \leq j \leq \lfloor |G_i|/I_i \rfloor$, $k \geq 0$. □

Lemma 3.2 provides a way for each region to directly compute the sequence of time points at which it has buckets to be merged. Based on this observation, we present the algorithm that given time t returns the set of regions, which have buckets to be merged at time t.

Algorithm for Fast Bucket Merging. Our implementation of the algorithm uses a hash table T to store the set of buckets that need to be merged at timestamp t. In particular, t is hashed to the index of a table cell which stores the set of (i,t) pairs, such that region G_i has buckets to be merged at time t. Figure 5 shows procedure InitializeFindRegions() which first computes I_i using Lemma 3.1. It then uses Lemma 3.2 to fill in the earliest time at which region G_i will have buckets to be merged. At time t, FindRegions(t) (Figure 6) retrieves the set of regions that have buckets to be merged, and deletes those regions from the hash table. Then, for each returned region, we compute its next merging time using Lemma 3.2 and store the results into the corresponding hash table cells for the future lookup.

3.2.4 Time and Space Complexity

The time complexity depends on the sketch bucket that we chose and the decay function f given by the user.

Algorithm 3.4: INITIALIZEFINDREGIONS()

Initialize hash table T
$I_0 \leftarrow 1$;
for $1 \leq i \leq \beta - 1$
  do $I_i \leftarrow \lfloor |G_{i-1}|/I_{i-1} \rfloor I_{i-1}$
comment: From Lemma 3.1
for $1 \leq i \leq \beta - 1$
  do if $\lfloor |G_i|/I_i \rfloor \geq 2$
  then Insert $(i, b_i + 2I_i)$ into hash table T
comment: Compute when G_i first has mergable buckets

Figure 5: Routine to initialize hash table with merging times

Algorithm 3.5: FINDREGIONS(t)

$M \leftarrow \emptyset$
for each $(i,t) \in T$
comment: Region G has buckets to be merged at time t

do $\begin{cases} M \leftarrow M \cup \{i\} \\ \textbf{if } (t - b_i)/I_i \mod \lfloor |G_i|/I_i \rfloor = 0 \\ \quad \textbf{then } t' \leftarrow t + 2I_i \\ \quad \textbf{else } t' \leftarrow t + I_i \\ \textbf{comment: } \text{Find when } G_i \text{ next has mergable buckets} \\ \text{Insert } (i, t') \text{ into hash table } T \end{cases}$

return (M)
comment: set of regions with buckets to be merged at time t

Figure 6: FINDREGIONS(t) finds mergable regions at time t

Theorem 3.4. *The (amortized) time complexity of the algorithm per update in Figure 3 is $O(Q(M/n) + \log Q)$, where M is the total number of merges happened in processing the stream, and*

1. $Q = O\left(\frac{1}{\varepsilon^2} \log \frac{\beta}{\delta} \log n\right)$ *is the size of the sketch bucket in words in [12].*
2. $Q = O\left(\frac{1}{\varepsilon} \log \left(\frac{\varepsilon n}{\log n}\right)\right)$ *is the size of the sketch bucket in words in [5].*

Proof. The per update cost is dominated by: (1) inserting the new element into the bucket, which takes time sublinear in the size of the sketch bucket: $\log Q$. (2) merging buckets when necessary, which can be carried out in time linear in the size of the bucket data structure [12, 5], so the amortized time for merge per update is $O(Q(M/n))$ (3) Updating the hash table, which has to be done once for every merge that occurs, and takes constant time. Combining these costs leads to the stated time complexity. □

Time dependence on decay function f. As stated in Theorem 3.4, the time complexity depends on the value of M, which in turn is determined by the choice of decay function f, since it defines the size of each region in the WBMH and hence the sequence of bucket merges during the stream processing. We show the consequence for various broad classes of decay function:

- In the case of no decay ($f(x) \equiv 1$), the region G_0 is infinitely large, so the algorithm maintains only one bucket and therefore no bucket merges will happen, i.e., $M = 0$, giving the time cost $O(\log Q)$.
- For exponential decay functions $f(x) = 2^{-\alpha x}$, $\alpha > 0$, since all the regions have the same size $|G_i| = \left\lfloor \frac{1}{\alpha} \log_2 \left(1 + \frac{\varepsilon}{2}\right) \right\rfloor$, $0 \le i \le \beta$, no bucket merges will happen, i.e., $M = 0$, giving the time cost $O(\log Q)$.
- For all other decay functions, such as polynomial decay $f(x) = (x+1)^{-a}$, $a > 0$, many bucket merges can happen. For a synchronous stream, there can be at most n bucket merges, as each merge conceptually places two adjacent stream elements which were in different buckets in the same bucket. Thus, whatever the decay function, the total number of merges cannot be larger than the stream size n, i.e., $M \le n$. So the amortized time cost $O(Q)$.

The space complexity includes the space cost for the buckets in the histogram and the hash table. The space to represent each bucket depends on the choice of the bucket sketch.

Theorem 3.5. *The space complexity of the algorithm in Figure 2, 3 and 4 is $O(\beta(Z + \log n))$ bits, where*

1. $\beta = \left\lceil \log_{1+\varepsilon/2}(f(0)/f(n)) \right\rceil$
2. $Z = O\left(\frac{1}{\varepsilon^2} \log \frac{\beta}{\delta} \log n \log m\right)$ *is the size of the bucket sketch in bits in [12].*
3. $Z = O\left(\frac{1}{\varepsilon} \log m \log\left(\frac{\varepsilon n}{\log n}\right)\right)$ *is the size of the bucket sketch in bits in [5].*

Proof. The number of buckets used is at most 2β. For the randomized sketch designed in [12], in order to have a δ failure probability bound, by the union bound, we need to set the failure probability for each bucket to be $\delta/(2\beta)$, so we get $Z = O\left(\frac{1}{\varepsilon^2} \log \frac{\beta}{\delta} \log n \log m\right)$ (Lemma 11 in [12]). For the deterministic sketch designed in [5], $Z = O\left(\frac{1}{\varepsilon} \log m \log\left(\frac{\varepsilon n}{\log n}\right)\right)$ (Section 3.1 in [5]). The size of the hash table can be set to $O(\beta)$ cells, because each of the β regions occupies at most one cell. Each cell uses $O(\log n)$ bits of space to store the region's index and the region's next merge time. So all together, the total space cost is $O(\beta(Z + \log n))$. □

Space dependence on decay function f. As shown in Theorem 3.5, the space complexity depends on the decay function f, since it determines the number of regions (implicitly the number of buckets) in WBMH. We show the consequence for various broad classes of decay function:

- For exponential decay functions $f(x) = 2^{-\alpha x}$, $\alpha > 0$, we have $\beta = \alpha n \log_{1+\varepsilon/2} 2$ and therefore the space complexity is $O(n(\log m)\log n)$ bits. This means that this algorithm needs space linear in the input size.

- For polynomial decay functions $f(x) = (x+1)^{-a}$, $a > 0$, since $\beta = a\log_{1+\varepsilon/2} n$, the space complexity is sublinear, $O\left(\frac{a}{\varepsilon^3}\log^2 n \log m \log\frac{\beta}{\delta}\right)$ using the sketch of [12], and $O(\frac{a}{\varepsilon^2}\log n \log m \log(\varepsilon n/\log n) + \log^2 n)$ using the sketch of [5];

- In the case of no decay ($f(x) \equiv 1$), the region G_0 is infinitely large, so the algorithm maintains only one bucket, giving space cost $O(Z + \log n)$.

Intuitively the algorithm can approximate S_τ^f with a relative error bound using small space if f decays more slowly than the exponential decay. Further, the space decreases the "slower" that f decays, the limiting case being that of no decay. We complement this observation with the result that the DCS problem under exponential decay requires linear space in order to provide relative error guarantees.

3.2.5 Asynchronous Streams

So far our discussion of the algorithm for relative error has focused on the case of strictly synchronous streams, where the elements arrive in order of timestamps. In an asynchronous setting, a new element (v_1, w_1, t_1) may have timestamp $t_1 < t$ where t is the current time. But this can easily be handled by the algorithm described above: the new element is just directly inserted into the earlier bucket which is responsible for timestamp t_1. Meanwhile, at every clock tick t, if no new element with timestamp t is received, we can still maintain the WBMH in the way as if we received a new *dummy* stream element whose timestamp is t, but we do not insert the dummy element into the WBMH. In other words, we create a new sketch bucket for the dummy element when necessary but do not insert it into the sketch bucket (leaving the new sketch bucket empty), and merge all the sketch buckets determined. Therefore, WBMH can be maintained exactly in the same way as in the case where a strictly synchronous stream is processed in Figure 3. The accuracy and space guarantees do not alter, although the time cost is affected because for each new element, we need to find the right bucket to insert it.

Let Q, Z, M be the same as defined in Theorem 3.4 and 3.5. Let L denote the age of the oldest stream element.

Corollary 3.1. *The (amortized) time complexity of the algorithm per timestep for an asynchronous stream is $O(Q(M/L) + (n/L)(\log Q + \log\beta))$. The space complexity of the algorithm for an asynchronous stream is $O(\beta(Z + \log L))$ bits, where $\beta = \left\lceil \log_{1+\varepsilon/2}(f(0)/f(L)) \right\rceil$.*

Proof. Time complexity. Note that the number of sketch buckets only depends on the decay function and the timestamp range in the stream, and there are no more than $2\beta = 2\left\lceil \log_{1+\varepsilon/2}(f(0)/f(L)) \right\rceil$ sketch buckets. All the sketch buckets can be managed by a balanced binary search tree with the timestamp ranges of the buckets being the keys, so the time cost in finding the bucket for the insertion of a new element is

$O(\log \beta)$. Inserting a new element into a sketch bucket costs time $O(\log Q)$. So the amortized time for inserting elements into WBMH per timestep is $O((n/L)(\log Q + \log \beta)))$. Adding the amortized time cost $O(Q(M/L))$ in merging buckets per timestep, we get the stated time complexity.

Space complexity. The space cost includes the space usage by the sketch buckets $O(\beta Z)$ and the space usage by the hashtable $O(\beta \log L)$. Add them together, we get the stated space complexity. □

We note that in the case where the stream size is relatively much smaller than the timestamp range in the stream, the actual space cost by our algorithm will be much smaller than the (worst case) space complexity stated in the above theorem, because in that case most of the sketch buckets will either be empty or only have a few elements inserted.

4 Lower Bounds

This section shows large space lower bounds for finite decay or (super) exponential decay for DCC on strictly synchronous streams. Since DCC is a special case of DCS, and every synchronous stream is also an asynchronous stream, these lower bounds also apply to DCS on asynchronous streams.

4.1 Finite Decay

Finite decay, defined in Section 1.2, captures the case when after some age limit N, the decayed weight is zero.

Theorem 4.1. *For any finite decay function f with age limit N, any streaming algorithm (deterministic or randomized) that can provide an estimate $\widehat{C}^f_\tau$ such that $|\widehat{C}^f_\tau - C^f_\tau| < \varepsilon C^f_\tau$ for any τ given at query time for a stream of elements drawn from a universe of size m must require $\Omega(N \log(m/N))$ bits of space.*

Proof. The bound follows from the hardness of finding the maximum element within a sliding window on a stream of integers. Tracking the maximum within a sliding window of size N over a data stream needs $\Omega(N \log(m/N))$ bits of space, where m is the size of the universe from which the stream elements are drawn (Section 7.4 of [7]).

We show that if there exists an algorithm to approximate $\widehat{C}^f_\tau$, where f has age limit N, then there is an algorithm to find the maximum of the last N elements in R, using the same space. Let α denote the value of the maximum element in the last N elements of the stream. By definition, the decayed weights of the N most recent elements are positive, while all older elements have weight zero.

Note that C^f_τ is a non-increasing function of τ, so $C^f_\tau \geq C^f_\alpha$ for any $\tau < \alpha$. Further, so $C^f_\alpha > 0$, and $C^f_\tau = 0$ for $\tau > \alpha$. If C^f_τ can be approximated with a good relative error, then it is possible to distinguish between the cases $C^f_\tau > 0$ and $C^f_\tau = 0$, for each value of τ. By repeatedly querying the data structure for C^f_τ for different values of τ, we find a value τ^* such that $C^f_{\tau^*} > 0$ and $C^f_{\tau^*+1} = 0$. Then τ^* must be α, the maximum element of the last N elements. □

Since sliding window decay is a special case of finite decay, this shows that approximating C_τ^f with f being a sliding window decay function cannot be solved with relative error in sublinear space. This resolves an open problem identified in [1].

4.2 Exponential Decay

Exponential decay functions $f(x) = 2^{-\alpha x}$, $\alpha > 0$ are widely used in non-correlated time decayed steaming data aggregation. It is easy to maintain simple sums and counts under such decay efficiently [4]. However, in this section we will show that it is *not* possible to approximate C_τ^f with relative error guarantees using small space if m (the size of the universe) is large and f is exponential decay. This remains true for other classes of decay that are "faster" than exponential decay. We first present two natural approaches to approximate C_τ^f under an exponential decay function f, and analyze their space cost to show that each stores large amounts of information.

Algorithm I. Since tracking the sum under exponential decay can be performed efficiently using a single counter, we can just track the decayed correlated count for each distinct $v \in [m]$: $W_v^f = \sum_{e_i \in R | v_i = v} f(t - t_i)$, then $C_\tau^f = \sum_{v \geq \tau} W_v^f$. To ensure an good estimate for C_τ^f, each W_v^f must be tracked with sufficiently many bits of precision. One approach is that for each distinct $v \in [m]$ we maintain the timestamps of the last $\lceil \frac{1}{\alpha} \log_2 \frac{1}{\varepsilon} \rceil$ elements of the substream $R_v = \{v_i \in R | v_i = v\}$. From these timestamps, one can approximate W_v^f with a ε relative error bound, and hence C_τ^f can be approximated with an ε relative error bound. Each timestamp is $O(\log n)$ bits, so the total space cost is $O\left(m (\log n) \lceil \frac{1}{\alpha} \log \frac{1}{\varepsilon} \rceil\right)$ bits. $\square$

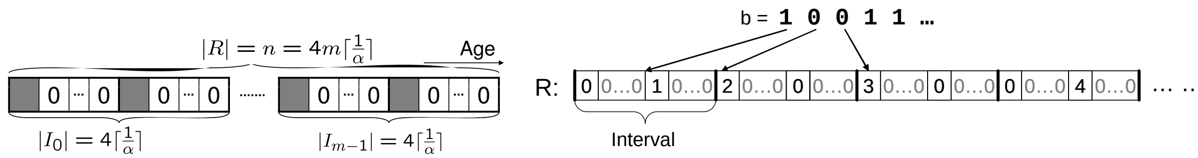
Algorithm II. The second algorithm tries to reduce the dependence on m by observing that for some close values of τ, the value of C_τ^f may be quite similar, so there is potential for "compression". As $f(x) = 2^{-\alpha x}$, $\alpha > 0$, we can write:

$$C_\tau^f = \sum_{v_i \geq \tau} 2^{\alpha(i-t)} = 2^{-\alpha t} \sum_{v_i \geq \tau} 2^{\alpha i}$$

where t is the query time. We reduce approximating C_τ^f with a relative error bound to a counting problem over an asynchronous stream with sliding window queries. We create a new stream R' in this model by treating each stream element as an item with timestamp set to its value v_i and with weight $2^{\alpha i}$. The query C_τ^f at time t can be interpreted as a sliding window query on the derived stream R' at time m with width $m - \tau + 1$. The answer to this query is $\sum_{v_i \geq \tau} 2^{\alpha i}$; by the above equation, scaling this down by $2^{\alpha t}$ approximates C_τ^f.

The derived stream R' can be summarized by sketches such as those in [12, 2]. These answer the sliding window query with relative error ε, implying relative error for C_τ^f. But the cost of these sketches applied here is $O(\frac{n}{\varepsilon} \log m)$ bits, linear in the stream length: in the reduction, the number of copies of each stream element increases exponentially, and the space cost of the sketches depends logarithmically on this quantity. $\square$

Hardness of Exponential Decay. Algorithm I is a conceptually simple approach, which stores information for each possible value in the domain. Algorithm II uses summaries that are compact in their original setting, but when applied to the DCC problem, their space must increase to give an accurate answer for any τ. The core reason for the high space cost of both algorithms is the fact that as τ varies between 0 and m, the value



(a) Setting the Intervals over a Stream

(b) Mapping from binary string b to intervals

Figure 7: Creating a stream for the lower bound proof using $p = 1$

of C_τ^f can vary over an exponentially large range, and a large data structure is required to track so many different values. This is made precise by the next theorem, which shows that the space cost of Algorithm I is close to optimal. We go on to provide a small space sketch with a weakened guarantee in Section 4.4, by limiting the range of values of C_τ^f for which an accurate answer is required.

Theorem 4.2. *For an exponential decay function $f(x) = 2^{-\alpha x}$, $\alpha > 0$ and $\varepsilon \le 1/2$, any algorithm (one-pass or multi-pass, deterministic or randomized) that provides $\widehat{C}_\tau^f$ over a stream of length $n = \Theta(m)$, such that $|\widehat{C}_\tau^f - C_\tau^f| < \varepsilon C_\tau^f$ for any τ given at query time must store $\Omega(m \log \frac{n}{m})$ bits, where m is the universe size.*

Proof. The proof uses a reduction from the INDEX problem in two-party communication complexity [10]. In the INDEX problem, Alice holds a binary string b of length N, and the second holds an index $i \in [N]$. Alice is allowed to send a single message to the second, who must then output the value of $b[i]$ (the ith bit of string b). Since no communication is allowed from Bob to Alice, the size of the message must be $\Omega(N)$ bits, even allowing the protocol a constant probability of failure [10].

We show that a small space streaming data structure to approximate DCC under exponential decay would allow a low communication complexity protocol for INDEX. Given a binary string b of length $N = mp$, we construct an instance of a stream, $R(b)$. Here m is the size of the domain of the stream values, and $p \ge 1$ is an integer parameter set later. The string b is divided into m partitions $\mathscr{P}_0, \mathscr{P}_1, \ldots, \mathscr{P}_{m-1}$, where $\mathscr{P}_i$ has bits $b[ip], b[ip+1], \ldots, b[(i+1)p-1]$.

Let $\ell = 2\lceil\frac{1}{\alpha}\rceil$. The stream $R(b)$ has $n = m2^p\ell$ elements. The n positions in $R(b)$ are divided into m intervals, $I_0, I_1, \ldots, I_{m-1}$, each of length $2^p\ell$, as shown in Figure 7(a) for the case $p = 1$; the more recent elements of the stream belong to the lower numbered interval. Each interval I_j is further divided into 2^p segments, each with ℓ elements. Each element is a tuple (v, i) where v is the value and i is the timestamp. The stream is synchronous, and the timestamps are consecutively increasing. The segments in I_j are numbered from 0 to $2^p - 1$, with the more recent segments in the stream getting smaller numbers. The value of every element of $R(b)$ is set to 0 except for m elements, one in each interval. The length p bit string in partition $\mathscr{P}_j$ in b is interpreted as an integer in the range $[0, 2^p - 1]$. In interval I_j, the segment numbered $\mathscr{P}_j$ is selected and the value of its most recent element is set to $j+1$, as shown in Figure 7(b) for the case $p = 1$.

Given a sketch that can approximate C_τ^f over $R(b)$ using ξ bits, we show a protocol for the INDEX problem with communication complexity ξ bits. Alice computes a sketch of $R(b)$, which can be used to approximate C_τ^f, and sends it to Bob. Bob, given an index i, computes $b[i]$ from the sketch, as follows. Let

$\tau = \lfloor \frac{i}{p} \rfloor$. Note that bit $b[i]$ lies in partition $\mathscr{P}_\tau$ in b. Bob recovers the complete integer $\mathscr{P}_\tau$, by using the sketch to distinguish between different assignments to the bit string $\mathscr{P}_\tau$.

Consider two different assignments to the string $\mathscr{P}_\tau$, representing two integers p_1 and p_2. Without losing the generality, let $p_1 > p_2$ (p_1 and p_2 cannot be the same). Let the value of C_τ^f for these two assignments be C_1 and C_2. Suppose that the sketch of $R(b)$ provides an answer for C_τ^f which has a relative error of 1/2 or less. Let $\widehat{C}_1$ and $\widehat{C}_2$ denotes the estimates returned for C_1, C_2 respectively. In Lemma 4.1, we show that if $p_1 > p_2$, then $C_1 < C_2$ and $\widehat{C}_1 < \widehat{C}_2$. Thus, all the estimates for C_τ^f over different assignments for $\mathscr{P}_\tau$ are in a total order. So, by using an estimate of C_τ^f, we can distinguish between different assignments to $\mathscr{P}_\tau$, and hence recover all of $\mathscr{P}_\tau$, and solve the INDEX problem. Thus the size of the sketch of $R(b)$ must be at least $\Omega(N) = \Omega(mp)$ bits.

The stream length is $n = \frac{m2^p}{\alpha}$, so the lower bound on the sketch size is $N = \Omega(m \log \frac{n}{m})$, for a constant α. Since the communication lower bound allows randomization, the space lower bound also holds for randomized stream algorithms. Since we did not assume that Alice processed the stream in one pass, this space lower bound holds even if the stream is allowed to be processed in multiple passes. □

Let p_1 and p_2, C_1 and C_2, and $\widehat{C}_1$ and $\widehat{C}_2$ be defined as in the proof of Theorem 4.2.

Lemma 4.1. *If $p_1 > p_2$, then $C_1 < C_2$ and $\widehat{C}_1 < \widehat{C}_2$.*

Proof. Since the sketch provides estimates that are within a relative error of $1/2$, we have:

$$\frac{C_1}{2} \leq \widehat{C}_1 \leq \frac{3C_1}{2} \tag{7}$$
$$\frac{C_2}{2} \leq \widehat{C}_2 \leq \frac{3C_2}{2} \tag{8}$$

Let c denote the current time.

$$C_\tau^f = \sum_{\{(v,i) \in R(b) | v \geq \tau\}} f(c-i)$$

For integer $j, 0 \leq j \leq m-1$, let the contribution of interval j to C_τ^f be defined as:

$$C(j) = \sum_{\{(v,i) \in I_j | v \geq \tau\}} f(c-i)$$

Also, since the two bit strings differ in the values assigned to $\mathscr{P}_\tau$, the corresponding streams differ in interval I_τ. Let $C_1(\tau)$ and $C_2(\tau)$ respectively denote the value of $C(\tau)$ for the two inputs. We note that $C_1(\tau)$ and $C_2(\tau)$ differ by a factor of at least 4, since they both contain an element with the same value in I_τ, but in different positions, so that the decayed weights differ by a factor of at least 4. Since $p_1 > p_2$, according to the stream construction in the proof of Theorem 4.2, we have

$$C_1(\tau) \leq \frac{C_2(\tau)}{4} \tag{9}$$

In Lemma 4.2, we show that C_τ^f is dominated by the term $C(\tau)$. More precisely:

$$C(\tau) < C^f_\tau < \frac{4}{3}C(\tau) \tag{10}$$

Combining Inequalities 9 and 10, we get

$$C_1 < \frac{4}{3}C_1(\tau) \le \frac{4}{3}\frac{1}{4}C_2(\tau) = \frac{1}{3}C_2(\tau) < \frac{1}{3}C_2 < C_2$$

Combining Inequalities 7, 8, and 9, we get:

$$\widehat{C}_1 \le \frac{3}{2}C_1 < \frac{3}{2}\frac{4}{3}C_1(\tau) = 2C_1(\tau) \le \frac{1}{2}C_2(\tau) < \frac{1}{2}C_2 \le \widehat{C}_2$$

□

Lemma 4.2.

$$C(\tau) < C^f_\tau < \frac{4}{3}C(\tau)$$

Proof. Note that $C^f_\tau = \sum_{j=0}^{m-1} C(j)$. For every $0 \le j < \tau$, we note that I_j does not have any tuples (v,i) with $v \ge \tau$. Thus, the contribution of such tuples to C^f_τ is 0, and so:

$$C^f_\tau = \sum_{j=\tau}^{m-1} C(j)$$

Also, for $j > \tau$, the contribution of $C(j)$ to C^f_τ is non-zero, since for $j > 0$, I_j has tuples (v,i) with $v \ge \tau$. Thus, $C(\tau) < C^f_\tau$, which proves one part of the Lemma.

Next, we note that for any integer ζ, no matter what the contents of $I_{\tau+\zeta}$ is,

$$C(\tau+\zeta) \le \frac{C(\tau)}{4^\zeta}$$

The reason is as follows. Note that $C(\tau)$ is non-zero, since there is at one element in I_τ with value greater than or equal to τ. Further, there is exactly one non-zero element in each interval $I_{\tau+\zeta}$. Since the difference in timestamps between the non-zero element in I_τ and $I_{\tau+\zeta}$ is at least $\zeta \cdot \ell$, we have:

$$C(\tau+\zeta) \le C(\tau) \cdot f(\ell\zeta) = C(\tau) \cdot 2^{-\alpha\ell\zeta} \le \frac{C(\tau)}{4^\zeta}$$

where the last step follows from the definition of ℓ.

$$C^f_\tau = \sum_{j=\tau}^{m-1} C(j) = \sum_{\zeta=0}^{m-1-\tau} C(\tau+\zeta) \le \sum_{\zeta=0}^{m-1-\tau} \frac{C(\tau)}{4^\zeta} < \sum_{\zeta=0}^{\infty} \frac{C(\tau)}{4^\zeta} = \frac{4}{3}C(\tau)$$

□

4.3 Super-exponential Decay

The result in Theorem 4.2 also holds for the super-exponential decay functions.

Theorem 4.3. *Consider a stream of length $n = \Theta(m)$, and a super-exponential decay function f. Any algorithm (one-pass or multi-pass, deterministic or randomized) that provides $\widehat{C}^f_\tau$, an estimate of C^f_τ, such that $|\widehat{C}^f_\tau - C^f_\tau| < \varepsilon C^f_\tau$ for any τ given at query time must store $\Omega\left(m \log \frac{n}{m}\right)$ bits, where m is the universe size.*

Proof. The proof is nearly identical to the one for Theorem 4.2, having the same structure and using the same reduction to the INDEX problem.

Again, Alice has a bit string b of length mp, which is divided into m partitions: $\mathscr{P}_0, \mathscr{P}_1, \ldots, \mathscr{P}_{m-1}$, where $\mathscr{P}_i$ has bits $b[ip], b[ip+1], \ldots, b[(i+1)p-1]$. Based on the bit string b, Alice creates a stream $R(b)$ of length $n = 2^p m\ell + c$, where $\ell = \lceil \log_\sigma 4 \rceil$ and c is the constant in the definition of super-exponential decay in Section 1.2, as follows. Each element is a (v,t) pair and elements are received in the order of their timestamps $0, 1, \ldots, 2^p m\ell + c - 1$, i.e., the stream $R(b)$ is strictly synchronous. The c elements with largest timestamps in $R(b)$ are assigned with value 0. Let $R'(b)$ denote the other elements in $R(b)$. For stream $R'(b)$, Alice assigns the values in the same way as she did for $R(b)$ in the proof for Theorem 4.2: (1) $R'(b)$ is divided into m intervals, $I_0, I_1, \ldots, I_{m-1}$, each of length $2^p\ell$; (2) Each interval I_j is further divided into 2^p segments, each with ℓ elements. The segments in I_j are numbered from 0 to $2^p - 1$, with the more recent segments in the stream getting smaller numbers; (3) The value of every element of $R'(b)$ is set to 0 except for m elements, one in each interval. The length p bit string in partition $\mathscr{P}_j$ in b is interpreted as an integer in the range $[0, 2^p - 1]$. In interval I_j, the segment numbered $\mathscr{P}_j$ is selected and the value of its most recent element is set to $j+1$.

Alice process the stream $R(b)$ and sends the sketch to Bob. Given the index i, Bob sets $\tau = \lceil \frac{i}{p} \rceil$ and queries the sketch for C^f_τ. Since the ages of the elements in $R'(b)$ are at least c, by the definition of super-exponential decay, any two neighboring elements in $R'(b)$ have their weights differ by a factor of at least σ. Thus, the two most recent elements in any two neighboring segments differ in their weights by a factor of at least 4, since we set $\ell = \lceil \log_\sigma 4 \rceil$. Further, since the c most recent elements in $R(b)$ will not have any contribution in C^f_τ, for any $\tau > 0$, because they are all assigned with value 0, we now have the same argument between the bit string b and stream $R'(b)$ as we did for the bit string b and stream $R(b)$ in the proof for Theorem 4.2: by using $\widehat{C}^f_\tau$, the estimate of C^f_τ returned by the sketch, Bob can reveal the value of $b[i]$. So the space cost for processing stream $R(b)$ of length $n = 2^p m\ell + c$ is at least mp bits. By replace p, we get the space lower bound of $\Omega\left(m \log \frac{n}{m}\right)$ bits, by constants of c and σ

Since the communication lower bound allows randomization, the space lower bound also holds for randomized stream algorithms. Since we did not assume that Alice processed the stream in one pass, this space lower bound holds even if the stream is allowed to be processed in multiple passes.

□

4.4 Finite (Super) Exponential Decay

As noted above, the lower bound proof relies on distinguishing a sequence of exponentially decreasing possible values of the DCC. In practical situations, it often suffices to return an answer of zero when the true answer is less than some specified bound μ. This creates a "finite" version of exponential decay.

Definition 4.1. *A decay function f is a finite exponential decay function with threshold μ, $0 < \mu < 1$, if: (1) $f(x) = 2^{-\alpha x}$, $\alpha > 0$, if $x \leq \frac{1}{\alpha}\log_2 \frac{1}{\mu}$ (which implies $f(x) \geq \mu$); (2) $f(x) = 0$, otherwise.*

Since finite exponential decay is a finite decay, the space lower bound in Theorem 4.1 implies space of $\Omega(\frac{1}{\alpha}\log\frac{1}{\mu})$ bits is necessary to approximate C_τ^f. A simple algorithm for C_τ^f simply stores all the stream elements with non-zero decayed weights. The space is $O\left(\frac{1}{\alpha}(\log m)\log\frac{1}{\mu}\right)$ bits, which is (nearly) optimal (treating $\log m$ as a small constant). This approach extends to the finite versions of super-exponential decay.

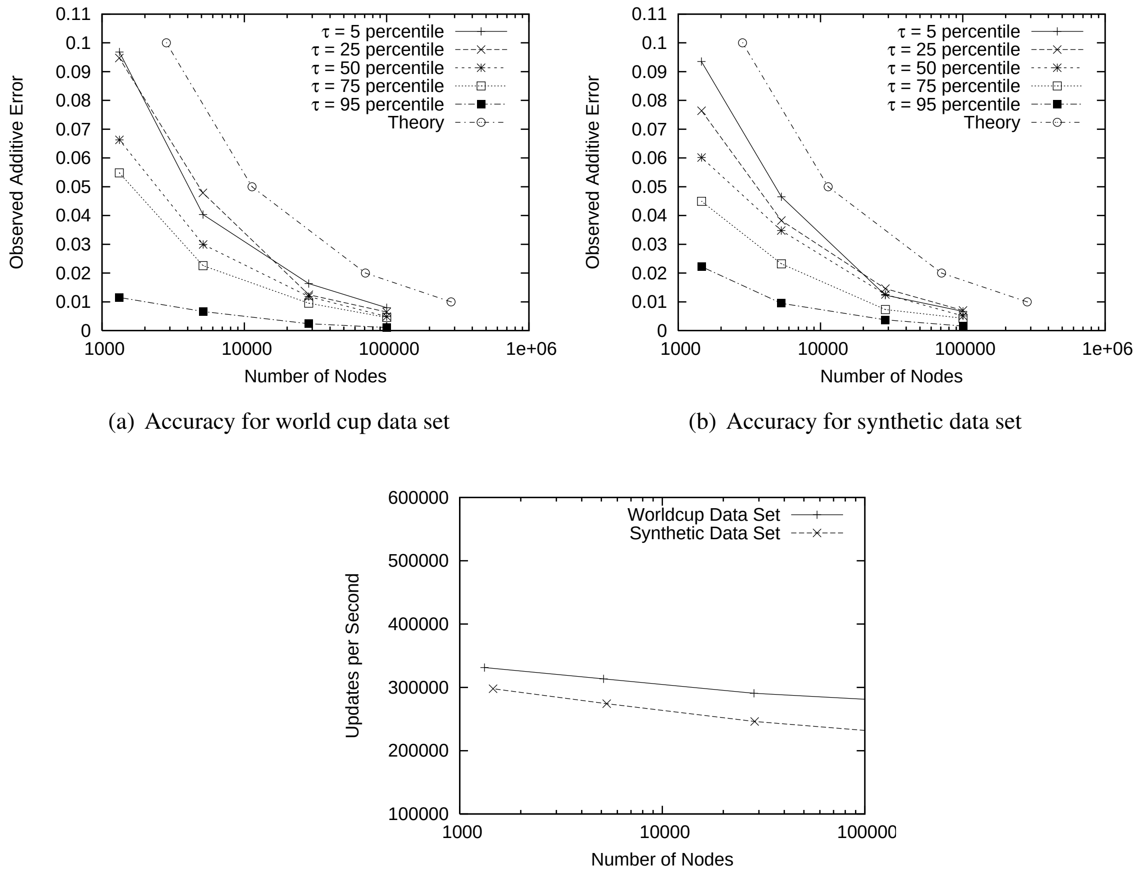
4.5 Sub-exponential decay

For any decay function $f(x)$, where $\lim_{x\to\infty} f(x) = 0$, we can always find $2^p m$ positions (timestamps) in the stream: $0 \leq x_1 < x_2 < \ldots < x_{2^p m}$, such that for every i, $1 < i \leq 2^p m$, we have $f(t-x_{i-1})/f(t-x_i) \leq \frac{1}{4}$. Thus, it is natural to analyze what happens when we apply the construction from the lower bound in Theorem 4.2 to streams under such functions. Certainly, the same style of argument constructs a stream that forces a large data structure. If we fix some m and set $p = 1$, the stream has to be truly enormous to imply a large space lower bound: e.g., for the polynomial decay function $f(x) = (x+1)^{-a}$, $a > 0$, we need $n = \Theta(2^{m/a})$ to force $\Omega(m)$ space. This is in agreement with the upper bounds in Section 3.2 which gave algorithms that depend logarithmically on n: for such truly huge values of n, this leads to a requirement of $\log 2^{m/a} = \Omega(m)$, so there is no contradiction.

5 Experiments

We present results from an experimental evaluation of the algorithms on two data sets. The first was web traffic logs from the 1998 World Cup on June 19th (the 'worldcup' data set) from `http://ita.ee.lbl.gov/`. Each stream element was a tuple (v, w, t), where v was the client id , w the packet size (modulo 100, simply to have initial weights bounded within a range), and t the timestamp, of the original web traffic logs, respectively. The dataset had 33695769 elements. The second was a synthetically generated data set (the 'synthetic' data set). The size of the synthetic data is the same as the worldcup data set. Here, the timestamp of an element is a random number chosen uniformly from the range $[1, \max_t]$ where $\max_t = 898293600$ is the maximum timestamp in the world cup data set. The value v is chosen uniformly from the range $[1, \max_v]$, where $\max_v = 1823218$ is the maximum value in the worldcup data set. The weight is chosen similarly, i.e. uniformly from the range $[1, \max_w]$ where $\max_w = 99$ is the maximum weight in the world cup data.

We implemented our algorithms using C++/STL and all experiments were performed on a SUSE Linux Laptop with 1GB memory. Both input streams were asynchronous, and elements do not arrive in timestamp



(a) Accuracy for world cup data set

(b) Accuracy for synthetic data set

(c) Throughput versus space on both data sets

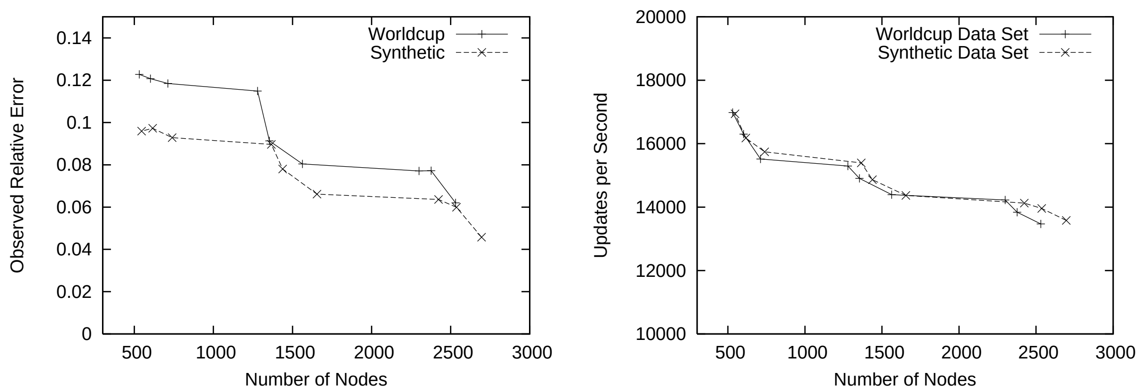
Figure 8: Throughput and accuracy with sliding window decay, additive error.

order.

Additive Error. We implemented the algorithm for additive error (Section 3.1) using the sketch in [6] as the basis. Note that the sketch in [6] provides the additional property of duplicate insensitivity, i.e., a re-insertion of the data into the sketch does not change the state of the sketch. Since our stream model does not have duplicates, our implementation of the sketch in [6] does not need to support duplicates detection and therefore improves the time efficiency in the stream processing.

Since a query for S_τ^f where f is not a sliding window decay can be reduced to queries for sliding window decay based DCS [4], we conducted experiments with the correlated sum S_τ^f where f is the sliding decay function. The window size is $4.5 \cdot 10^7$ for the synthetic data and 3600 for the worldcup data. We tried a range of values of the threshold τ, from the 5 percent quantile (5th percentile) of the values of stream elements to the 95 percent quantile. We analyzed the accuracy of the estimates returned by the sketch, for a given space budget.

Figures 8(a) and 8(b) show the observed additive error as a function of the space used by the algorithm



(a) Relative Error versus Space

(b) Throughput versus Space

Figure 9: Performance of Relative Error Algorithm, with Polynomial Decay.

for different values of τ. The space cost is measured in the number of nodes, where each node is the space required to store a single stream element (v,w,t), which takes a constant number of bytes. This cost can be compared to the naive method which stores all input elements (nearly 34 million nodes). The observed error is usually significantly smaller than the guarantee provided by theory. The theoretical guarantee holds irrespective of the value of τ or the window size. Note that the additive error decreased as the square root of the space cost, as expected. Figure 8(c) shows the throughput, which is defined as the number of stream elements processed per second, as a function of the space used. From the results, the trend is for the throughput to decrease slowly as the space increases. Across a wide range of values for the space, the throughput is between 250K and 350K updates per second.

Relative Error. We implemented WBMH and the sketch designed in [12] as the bucket sketch embedded in WBMH. We performed similar experiments to test our algorithms for relative error, based on the polynomial decay function $f(x) = 1/(x+1)^{1.5}$, a non-exponential converging decay. The thresholds are the same as in the additive error algorithm. The results are shown in Figure 9. In general, the space cost for a given error for polynomial decay was much smaller than the algorithm for sliding windows (Figure 9(a)). This greater space efficiency comes at some cost: we have to fix the decay function *a priori*—the additive error result allows the decay function to be specified at query time. The throughput for the relative error algorithm is also appreciably lower than the additive error algorithm (Figure 9(b)), by over an order of magnitude. This is partly due to the greater time complexity of the relative error algorithm caused by the periodic bucket merging operations which access every node in the merged buckets, and partly because our implementation is not fully tuned.

6 Concluding Remarks

Our results shed light on the problem of computing correlated sums over time-decayed streams. The upper bounds are quite strong, since they apply to asynchronous streams with arbitrary timestamps. It is also possible to extend these results to a distributed streaming model, since the summarizing data structures used can naturally be computed over distributed data, and merged together to give a summary of the union of the streams. The lower bounds are similarly strong, since they apply to the most restricted model, for computing DCC where there is exactly one arrival per time unit.

The correlated sum is at the heart of many correlated aggregates, but there are other natural correlated computations to consider which do not follow immediately from DCS. Some we expect to be hard in general: correlated maximum $\max_{v_i > \tau} w_i f(t - t_i)$ has a linear space lower bound under finite decay functions, since this lower bound follows from the uncorrelated case. Other analysis tasks seem feasible but challenging: for example, to output a good set of cluster centers for those points with $v_i > \tau$, weighted by $w_i f(t - t_i)$. It will be of interest to understand exactly which such correlated aggregations are possible in a streaming setting.

Acknowledgments. We thank Divesh Srivastava for helpful discussions, and Kewei Tu for useful pointers on Section 3.2.3.

References

[1] R. Ananthakrishna, A. Das, J. Gehrke, F. Korn, S. Muthukrishnan, and D. Srivastava. Efficient approximation of correlated sums on data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):569–572, 2003.

[2] C. Busch and S. Tirthapura. A deterministic algorithm for summarizing asynchronous streams over a sliding window. In *STACS*, 2007.

[3] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In *VLDB*, 1996.

[4] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, 2003.

[5] G. Cormode, F. Korn, and S. Tirthapura. Time-decaying aggregates in out-of-order streams. In *PODS*, 2008.

[6] G. Cormode, S. Tirthapura, and B. Xu. Time-decaying sketches for sensor data aggregation. In *PODC*, 2007.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[8] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD*, 2001.

[9] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, 2002.

[10] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[11] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys*, 2004.

[12] B. Xu, S. Tirthapura, and C. Busch. Sketching asynchronous data streams over sliding windows. *Distributed Computing*, 20(5):359–374, 2008.