Noname manuscript No.
(will be inserted by the editor)

Sketching Distributed Sliding-Window Data Streams

Odysseas Papapetrou · Minos Garofalakis · Antonios Deligiannakis

the date of receipt and acceptance should be inserted later

Abstract While traditional data-management systems focus on evaluating single, ad-hoc queries over static data sets in a centralized setting, several emerging applications require (possibly, continuous) answers to queries on *dynamic* data that is widely *distributed* and constantly updated. Furthermore, such query answers often need to discount data that is "stale", and operate solely on a *sliding window* of recent data arrivals (e.g., data updates occurring over the last 24 hours). Such *distributed data streaming* applications mandate novel algorithmic solutions that are both time- and space-efficient (to manage high-speed data streams), and also communication-efficient (to deal with physical data distribution). In this paper, we consider the problem of complex query answering over distributed, high-dimensional data streams in the sliding-window model. We introduce a novel sketching technique (termed *ECM-sketch*) that allows effective summarization of streaming data over both time-based and count-based sliding windows with probabilistic accuracy guarantees. Our sketch structure enables point as well as inner-product queries, and can be employed to address a broad range of problems, such as maintaining frequency statistics, finding heavy hitters, and computing quantiles in the sliding-window model. Focusing on distributed environments, we demonstrate how ECM-sketches of individual, local streams can be composed to generate a (low-error) ECM-sketch summary of the order-preserving merging of all streams; furthermore, we show how ECM-sketches can be exploited for continuous monitoring of sliding-window queries over distributed streams. Our extensive experimental study with two real-life data sets validates our theoretical claims and verifies the effectiveness of our techniques. To the best of our knowledge, ours is the first work to address efficient, guaranteed-error complex query answering over distributed data streams in the sliding-window model.

O. Papapetrou
Technical University of Crete
E-mail: papapetrou@softnet.tuc.gr

M. Garofalakis
Technical University of Crete
E-mail: minos@softnet.tuc.gr

A. Deligiannakis
Technical University of Crete
E-mail: adeli@softnet.tuc.gr

1 Introduction

The ability to process, in real time, continuous high-volume *streams* of data is a common requirement in many emerging application environments. Examples of such applications include, sensor networks, financial data trackers, and intrusion-detection systems. As a result, in recent years, we have seen a flurry of activity in the area of *data-stream processing*. Unlike conventional database query processing that requires several passes over a static, archived data image, data-stream processing algorithms often rely on building concise, approximate (yet, accurate) *sketch synopses* of the input streams in real time (i.e., in one pass over the streaming data). Such sketch structures typically require *small space and update time* (both significantly sublinear in the size of the data), and can be used to provide *approximate query answers* with guarantees on the quality of the approximation. These answers can be more than sufficient for typical exploratory analysis of massive data, where the goal is to detect interesting statistical behavior and patterns rather than obtain answers that are precise to the last decimal. Large-scale stream processing applications are also inherently *distributed*, with several remote sites observing their local stream(s) and exchanging information through a communication network. This distribution of the data naturally imposes critical *communication-efficiency* requirements that prohibit naïve solutions that centralize all the data, due to its massive vol-

ume and/or the high cost of communication (e.g., in sensornets). Communication efficiency is particularly important for *distributed event-monitoring* scenarios (e.g., monitoring sensor or IP networks), where the goal is real-time tracking of distributed measurements and events, rather than one-shot answers to sporadic queries [33].

Several query models for streaming data have been explored over the past decade. Streaming data items naturally carry a notion of "time", and, in many applications, it is important to be able to downgrade the importance (or, weight) of older items; for instance, in the statistical analysis of trends or patterns in financial data streams, data that is more than a few months old might be considered "stale" and irrelevant. Various *time-decay models* for querying streaming data have been proposed in the literature, mostly differentiating on the relation of an item's weight to its age (e.g., exponential or polynomial decay [7]). The *sliding-window* model [16] is one of the most prominent and intuitive time-decay models that considers only a window of the most recent items seen in the stream thus far (i.e., items outside the window are "aged out" or given a weight of zero). The window itself can be either *time-based* (i.e., items seen in the last N time units) or *count-based* (i.e., the last N items). Several algorithms have been proposed for maintaining different types of statistics over sliding-window data streams while requiring time and space that is significantly sublinear (typically, poly-logarithmic) in the window size [16,21,32,34]. Still, the bulk of existing work on the sliding-window model has focused on tracking basic counts and other simple aggregates (e.g., sums) over one-dimensional streams in a centralized setting. Recent work has also considered the case of distributed data; however, no existing techniques can handle flexible, complex aggregate queries over rapid, high-dimensional distributed data streams, e.g., with each dimension corresponding to the number of packets originating by an IP address, and the number of possible IP addresses reaching 2^{48} for IPv6.

Example: Recent work on effective network-monitoring systems (e.g., for detecting DDoS attacks or network-wide anomalies in large-scale IP networks) has stressed the importance of an efficient *distributed-triggering* functionality [26,28, 24,23]. In their early work, Jain et al. [26] discuss a generic distributed attack-detection scheme relying on the ability to maintain frequency statistics for high-dimensional data over sliding windows. In particular, each node (e.g., a network router implementing Cisco's Netflow protocol, a wireless access point, or a peer in a P2P network) maintains a sliding-window count of all observed messages for each target IP address. If this count exceeds a pre-determined threshold, which is determined based on the capacity of the target machine (possibly expressing the fair share of each client to the target machine), an event is triggered to a central coordinator as a warning of possible overloading. The coordinator then collects network-wide statistics to monitor overloaded nodes or abnormal behavior. More recent efforts have focused on different variants and extensions of this basic scheme, often requiring more extensive data/statistics collection and more sophisticated analyses [24,23]. (Note that such data collection mechanisms are supported by commercial products, such as the Cisco Netflow Collection Engine solution.)

The ability to efficiently summarize high-dimensional data over sliding windows is obviously crucial to such monitoring schemes, given the tremendous volume of network-data streams and their massive domain sizes (e.g., 2^{48} for IPv6 addresses). This raises a critical need for synopsis data structures that can compactly capture accurate frequency statistics for a vast domain space over sliding windows. Furthermore, to enable the coordinator to aggregate data coming from different nodes (a requirement for detecting DDoS attacks), we need to be able to *compose* individually constructed synopses to a single synopsis which can capture the global state of the network and help isolate network-wide abnormalities. Thus, we are faced with the difficult challenge of designing effective, composable synopses that can support potentially complex sliding-window analysis queries over massive, distributed network-data streams. □

Note that similar requirements are frequently observed in other domains, e.g., for identifying misbehaving nodes in large wireless networks, for training of classifiers with distributed training data that expires over time, and for ranking products in a cloud-based e-shop, based on the number of recent visits of each product.

Our Contributions. In this paper, we consider the problem of answering potentially complex continuous queries over distributed, high-dimensional data streams in the sliding-window model. Our contributions can be summarized as follows.

• **ECM-Sketches for Sliding-Window Streams.** We introduce a novel sketch synopsis (termed *ECM-sketch*) that allows effective summarization of streaming data over both time-based and count-based sliding windows with probabilistic accuracy guarantees. In a nutshell, our ECM-sketch combines the well-known Count-Min sketch structure [11] for conventional streams with state-of-the-art tools for sliding-window statistics. The end result is a sliding-window sketch synopsis that can provide provable, guaranteed-error performance for point, as well as inner-product, queries, and can be employed to address a broad class of queries, such as maintaining frequency statistics, finding heavy hitters, and computing quantiles in the sliding-window model.

• **Time-based Sliding Windows over Distributed Streams.** Focusing on distributed environments, we demonstrate how ECM-sketches summarizing time-based sliding windows of individual streams can be composed to generate a guaranteed-error ECM-sketch synopsis of the order-preserving merg-

ing of all streams. While conventional Count-Min sketches are trivially composable, composing ECM-sketches is more challenging since it requires merging of the sliding-window statistics maintained in the sketch. Therefore, as part of our merging solution for ECM-sketches, we also provide the theoretical foundations and an efficient algorithm for merging sliding window statistics of deterministic algorithms [16, 21]. This is an important result on its own given the wide applicability of these algorithms, as well as their substantially higher efficiency and compactness compared to randomized sliding window algorithms, which are more easily composable [21,35]. This increased efficiency comes at the cost of a slight inflation of the worst-case error guarantee due to the composition, which however can be easily controlled, even in large hierarchical networks with iterative mergings.

• **Continuous Query Monitoring for Complex Queries over Distributed Streams.** We show how ECM-sketches can be exploited in the context of the geometric framework of Sharfman et al. [33] for continuous monitoring of sliding-window queries over distributed streams. We demonstrate the sketch-enhanced geometric framework by addressing two frequent requirements of distributed stream monitoring applications: (a) maintaining the set of items with a frequency surpassing a threshold (e.g., the IP addresses that exchange an excessive amount of messages over a sliding window), and, (b) maintaining an estimate for the self-join size of a stream over the sliding window, a useful measure for constructing efficient distributed query execution plans. Empowered by the compactness and efficiency of the underlying sketches, the geometric framework can now monitor such queries in a both computational-efficient and network-efficient manner.

• **Experimental Study and Validation.** We perform a thorough experimental evaluation of our techniques using two massive real-life data sets, in both centralized and distributed settings. The results of our study verify the efficiency and effectiveness of our ECM-sketch synopses in a variety of applications, and expose interesting functional trade-offs. When compared to algorithms based on randomized sliding window synopses – which are the only ones that were considered for composition up to now – ECM-sketches reduce the memory and computational requirements by at least one order of magnitude with a very small loss in accuracy. Similar savings apply to the network requirements.

2 Related Work

Centralized and Distributed Data Streams. Most prior work on data-stream processing has focused on developing space-efficient, one-pass algorithms for performing a wide range of *centralized, one-shot computations* on massive data streams; examples include computing quantiles [22], estimating distinct values [19], counting frequent elements (i.e., "heavy hitters") [6, 10], and estimating join sizes and stream norms [1, 11]. Out of these efforts, flexible, general-purpose sketch summaries, such as the AMS [1] and the Count-Min sketch [11] have found wide applicability in a broad range of stream-processing scenarios. More recent efforts have also concentrated on *distributed-stream* processing, proposing communication-efficient streaming tools for handling a number of query tasks, including distributed tracking of simple aggregates [30], quantiles [9], and join aggregates [8], as well as monitoring distributed threshold conditions [33]. All the above-referenced works assume a traditional, "full-history" data stream and do not address the issues specific to the sliding-window model.

Sliding-Window Stream Queries. As mentioned earlier, the bulk of existing work on the sliding-window model has focused on algorithms for maintaining simple statistics, such as basic counts and sums, in space and time that is significantly sub-linear (typically, poly-logarithmic) in the sliding-window size N. *Exponential histograms* [16] are a state-of-the-art deterministic technique for maintaining ϵ-approximate counts and sums over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space. *Deterministic waves* [21] solve the same basic counting/summation problem with the same space complexity as exponential histograms, but improve the worst-case update time complexity to $O(1)$; on the other hand, *randomized waves* [21] rely on randomization through hashing to track *duplicate-insensitive* counts (i.e., `COUNT-DISTINCT` aggregates) over sliding windows. While randomized waves can be easily composed (in distributed settings), they come with an increased space requirement of $O(\frac{\log(1/\delta)}{\epsilon^2}\log^2 N)$, where δ is a small probability of failure. Xu et al. [35] describe a randomized, sampling-based synopsis, very similar to randomized waves, for tracking sliding-window counts and sums with out-of-order arrivals (e.g., due to network delays) in a distributed setting. As with randomized waves, their space requirements are also quadratic in the inverse approximation error; furthermore, their approach requires knowledge of the maximum number of elements in any sliding window (to set up the synopsis data structure), which could be problematic in dynamic, widely-distributed environments. Cormode et al. [14] also propose randomized techniques for handling out-of-order arrivals for tracking duplicate-insensitive sliding window aggregates. To address the high cost associated with randomized data structures, Busch and Tirthapura propose a deterministic structure for handling out-of-order arrivals in sliding windows [3]. Similar to the other deterministic structures, this structure also does not allow composition and focuses only on basic counts and sums.

More recent works develop protocols for efficient continuous monitoring of sliding window aggregates over distributed architectures [5, 12, 13, 15]. These techniques typically focus on reducing the network requirements for main-

taining random samples or simple statistics (such as basic counts, heavy hitters, and quantiles) with accuracy guarantees. Some aspects of these techniques could find use in the case of ECM-sketches as well. In this work we have selected to build the continuous monitoring scheme over the geometric method. The geometric method goes beyond monitoring simple linear aggregates, by enabling distributed monitoring of (possibly) complex functions that can be expressed over the average values of the monitored variables, e.g., self-join and inner product sizes. As such, we are able to monitor any function that can be supported by the ECM-sketch.

Going beyond counts, sums, and simple aggregates, there is surprisingly little work in the more general problem of maintaining general, frequency-distribution synopses over high-dimensional streaming data in the sliding-window model. Hung and Ting [25] and Dimitropoulos et al. [17] propose synopses based on Count-Min sketches for tracking heavy hitters and frequency counts over sliding windows; still, their techniques rely on keeping simple equi-width counters within the sketch, and, thus, cannot provide any meaningful error guarantees, especially for small query ranges. Similarly, the hybrid histograms of Qiao et al. [32] combine exponential histograms with simplistic equi-width histograms for answering sliding-window range queries; again, these structures cannot give meaningful bounds on the approximation error and cannot be composed in a distributed setting.

Chakrabati et al. briefly sketched the combination of Count-min sketches and exponential histograms for computing the entropy of a stream over a sliding window [4]. Compared to that work, our work goes several steps forward. First, we provide important materialization details, which were not discussed in [4]. For example, we show how to automatically choose the sketch configuration that satisfies the accuracy requirements and minimizes space complexity. Second, we present merging algorithms for ECM-sketches (even the ones that are based on deterministic sliding window algorithms), which are necessary in many domains involving distributed stream processing. Finally, we present algorithms for distributed continuous monitoring using ECM-sketches.

An early version of this work has previously appeared in [31]. Compared to [31], in this article we follow a more rigorous analysis, which leads to tighter theoretical error bounds, and to substantial reduction of the size of the sketch. Sketch size is typically reduced by a factor of three for ECM-sketches based on deterministic sliding window algorithms, and by a factor of six for the ones based on randomized algorithms. Furthermore, we elaborate on continuous function monitoring with ECM-sketches, which was only briefly mentioned in the original paper. This elaboration includes a novel efficient monitoring algorithm, accompanied with proof of correctness, and with extensive experimental evaluation.

3 Preliminaries

ECM-sketches combine the functionalities of Count-Min sketches [11] and exponential histograms [16]. We now describe the two structures, focusing on the aspect related to our work.

Count-Min Sketches. Count-Min sketches are a widely applied sketching technique for data streams. A Count-Min sketch is composed of a set of d hash functions, $h_1(\cdot)$, $h_2(\cdot)$, ..., $h_d(\cdot)$, and a 2-dimensional array of counters of width w and depth d. Hash function h_j corresponds to row j of the array, mapping stream items to the range of $[1 \dots w]$. Let $CM[i,j]$ denote the counter at position (i,j) in the array. To add an item x of value v_x in the Count-Min sketch, we increase the counters located at $CM[j,h_j(x)]$ by v_x, for $j \in [1 \dots d]$. A point query for an item q is answered by hashing the item in each of the d rows and getting the minimum value of the corresponding cells, i.e., $\min_{j=1}^{d} CM[j,h_j(q)]$. Note that hash collisions may cause estimation inaccuracies – only overestimations. By setting $d = \lceil \ln(1/\delta) \rceil$ and $w = \lceil e/\epsilon \rceil$, where e is the base of the natural logarithm, the structure enables point queries to be answered with an error of less than $\epsilon ||a||_1$, with a probability of at least $1-\delta$, where $||a||_1$ denotes the number of items seen in the stream. Similar results hold for range and inner product queries.

Exponential Histograms. Exponential histograms [16] are a deterministic structure, proposed to address the basic counting problem, i.e., for counting the number of true bits in the last N stream arrivals. They belong to the family of methods that break the sliding window range into smaller windows, called buckets or basic windows, to enable efficient maintenance of the statistics. Each bucket contains the aggregate statistics, i.e., number of arrivals and bucket bounds, for the corresponding sub-range. Buckets that no longer overlap with the sliding window are expired and discarded from the structure. To compute an aggregate over the whole (or a part of) sliding window, the statistics from all buckets overlapping with the query range are aggregated. For example, for basic counting, aggregation is a summation of the number of true bits in the buckets. A possible estimation error can be introduced due to the oldest bucket inside the query range, which usually has only a partial overlap with the query. Therefore, the maximum possible estimation error is bounded by the size of the last bucket.

To reduce the space requirements, exponential histograms maintain buckets of exponentially increasing sizes. Bucket boundaries are chosen such that the ratio of the size of each bucket b with the sum of the sizes of all buckets more recent than b is upper bounded. In particular, the following invariant (*invariant 1*) is maintained for all buckets j: $C_j/(2(1+\sum_{i=1}^{j-1} C_i)) \leq \epsilon$ where ϵ denotes the maximum acceptable relative error and C_j denotes the size of bucket j (number

Notation	Description
N	Length of the sliding window, in time units or in number of arrivals
$h_i(\cdot)$	Hash function i of the Count-Min sketch
a_r, b_r	Substream of stream a, b, within the query range r
$f_a(x,r)$	Frequency of item x in stream a, within the query range r
$E_a(i,j,r)$	Estimated value of the ECM-sketch counter for stream a in position (i,j) for query range r
$a_r \odot b_r, \widehat{a_r \odot b_r}$	Real and estimated inner product of a_r and b_r
$u(N,S)$	Upper bound of number of arrivals on stream S within the sliding window of length N

Table 1 Frequently used notation.

of true bits arrived in the bucket range), with bucket 1 being the most recent bucket. Queries are answered by summing the sizes of all buckets that fully overlap the query range, and half of the size of the oldest bucket, if it partially overlaps the query. The estimation error is solely contained in the oldest bucket, and is therefore bounded by this invariant, resulting to a maximum relative error of ϵ.

4 ECM-Sketches

We now describe ECM-sketches (short for Exponential Count-Min sketches), a composable sketch for maintaining data stream statistics over sliding windows in distributed environments. ECM-sketches combine the functionality of Count-Min sketches and sliding windows, and support both time-based and count-based sliding windows under the cash register model. Therefore, they can be used for compactly summarizing high-dimensional streams over sliding windows, i.e., to maintain the observed frequencies of the stream items within the sliding window range.

The core of the structure is a modified Count-Min sketch. Count-Min sketches alone cannot handle the sliding window requirement. To address this limitation, ECM-sketches replace the Count-Min counters with sliding window structures. Each counter is maintained as a sliding window, covering the last N time units, or the last N arrivals, depending on whether we need time-based or count-based sliding windows.

As discussed in Section 2, there have been several algorithms proposed for sliding window maintenance. Due to the large expected number of sliding window counters in ECM-sketches, we require an algorithm with a small memory footprint. Existing randomized algorithms for sliding window synopses (as discussed in Section 2) appear to have a quadratic dependence to ϵ and are therefore not good for our purposes. Instead, we employ exponential histograms, a compact and efficient deterministic synopsis [16]. Each of the Count-Min counters is implemented as an exponential

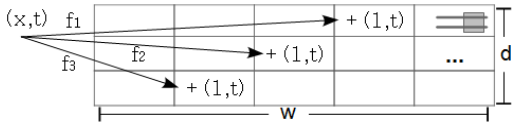

Fig. 1 Adding an element to the ECM-sketch.

histogram, configured to provide an ϵ approximation for any query within a sliding window of length N, i.e., the estimation $\hat{x}$ of the counter for any query range within the sliding window length is in the range of $(1 \pm \epsilon)x$ of the true value x of the counter. We will be discussing our choice for exponential histograms again in more detail in the following section, where we will consider alternative deterministic and randomized algorithms.

Adding an item x to the structure is similar to the case of standard Count-Min sketches. The process for time-based sliding windows is depicted in Figure 1. First, the counters $CM[j, h_j(x)]$, where $j \in \{1 \dots d\}$, corresponding to the d hash functions are detected. For each of the counters, we register the arrival of the item at time t, and remove all expired information, i.e., the buckets of the exponential histogram that have no overlap with the sliding window range. The process for count-based sliding windows is similar, but instead of registering each arrival with system time t, we register it with the count of arrivals since the beginning of the stream.

The challenges that need to be addressed for the integration of exponential histograms with Count-Min sketches are: (a) to take into account the additional error introduced by the sliding window counters for deriving the accuracy guarantees for ECM-sketches (presented in the remainder of this section), and, (b) to enable composition of a set of ECM-sketches to a single ECM-sketch representing the order-preserving merging of the corresponding individual streams (Section 5).

4.1 Query Answering

We now explain how ECM-sketches support point queries, inner product and self-join size queries, and derive probabilistic guarantees for the estimation accuracy. Our analysis covers both sliding window models, i.e., count-based and time-based.

Point Queries. A point query (x, r) is a combination of an item identifier x, and the query range r defined either as number of time units or number of arrivals. Point queries are executed as follows. The query item is hashed to the d counters $CM[j, h_j(x)]$ where $(j \in \{1 \dots d\})$, and the estimate of each counter $E(j, h_j(x), r)$ for the query range is computed. The estimate value for the frequency of x is $\hat{f}(x,r) = \min_{j=1 \dots d} E(j, h_j(x), r)$.

Let δ_{cm} and ϵ_{cm} denote the configuration parameters of the Count-Min sketch, whereas ϵ_{sw} denotes the configuration parameter of the exponential histogram. With $||a_r||_1$ we denote the number of arrivals within the query range. The following theorem provides probabilistic guarantees for the approximation quality for point queries and enables optimally setting ϵ_{cm} and ϵ_{sw}. As is typical for small-space sketches, the error guarantees are relative to the stream characteristics, i.e., the L1 norm.

Theorem 1 *For any ϵ within $(0,1)$, an ECM-sketch constructed with $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$ and $\epsilon_{sw} = \epsilon$ satisfies $Pr[|\hat{f}(x,r) - f(x,r)| \leq \epsilon||a_r||_1] \geq 1 - \delta_{cm}$. Furthermore, the aforementioned combination of ϵ_{cm} and ϵ_{sw} minimizes the space complexity of the sketch.*

Proof Special case of Theorem 3, with $\delta_{sw} = 0$. □

Inner Product and Self-Join Size Queries. Another frequent query type is the cardinality of the inner product. Given two streams a and b, the inner product is defined as $a \odot b = \sum_{x \in \mathcal{D}} f_a(x) \times f_b(x)$, where $\mathcal{D}$ denotes the input domain, i.e., the distinct input elements, and $f_a(x)$ (resp. $f_b(x)$) denotes the frequency of element x in stream a (resp. stream b). Self-join size queries, also called the second frequency moment F_2, are a special case of inner product queries defined over a single stream: $F_2(a) = \sum_{x \in \mathcal{D}} (f_a(x))^2$. Both inner product and self-join size queries are very important for databases, e.g., for building query execution plans, and they can be efficiently and accurately estimated for streams in both the cash register and turnstile model [29]. However, similar to point queries, computing these queries over sliding windows is challenging.

ECM-sketches can be used to address this type of queries as well. Let a_r (resp., b_r) denote the substream of stream a (resp., b) within the query range. With CM_a we denote the corresponding ECM-sketch for stream a_r, and with $E_a(i,j,r)$ we denote the estimated value of the counter of CM_a in position (i,j), for query range r. Also, $f_a(x,r)$ and $\hat{f}_a(x,r)$ denote the real and estimated frequency of x in stream a_r.

The inner product of two streams a and b in a range r is defined as $a_r \odot b_r = \sum_{x \in \mathcal{D}} f_a(x,r) f_b(x,r)$. Using the ECM-sketches of a and b, we estimate it as follows: $\widehat{a_r \odot b_r} = \min_j(\widehat{a_r \odot b_r})_j$, where $(\widehat{a_r \odot b_r})_j = \sum_{i=1}^{w} E_a(i,j,r) \times E_b(i,j,r)$. The following theorem bounds the approximation error.

Theorem 2 *For any ϵ within $(0,1)$, two ECM-sketches constructed with $\epsilon_{cm} = \epsilon/(\epsilon+1)$ and $\epsilon_{sw} = \sqrt{\epsilon+1} - 1$ satisfy $Pr[|\widehat{a_r \odot b_r} - a_r \odot b_r| \leq \epsilon||a_r||_1||b_r||_1] \geq 1 - \delta_{cm}$. Furthermore, the aforementioned combination of ϵ_{cm} and ϵ_{sw} minimizes the space complexity of the sketches.*

Proof In the appendix.

Time-based ECM-Sketches. Exponential histograms were originally developed for count-based sliding windows (e.g., count the number of true bits in the last 100 arrivals), but they can be extended for time-based sliding windows as well (e.g., count the number of true bits arriving in the last 1000 sec.). Our solution can handle concurrent bit arrivals as well as arrivals at arbitrary rates, and similar to the count-based histograms, its memory footprint (the number of buckets) scales logarithmically with the number of arrivals within the sliding window. First, each entry in the data structure is identified using its arrival time, instead of using its position in the stream. To reduce memory, arrival times are stored in wraparound counters of $O(\ln N)$ bits, where N is the length of the sliding window, e.g., in milliseconds. Second, entries expire based on their arrival time, and not on their position in the stream. Finally, we require an upper bound of the number of arrivals within the sliding window time range for each stream S, denoted as $u(N,S)$. Note that this is required only for computing the maximum memory requirements of the structure a priori; it does not have an impact on the actual required memory or quality of ECM-sketches. Furthermore, the bound can be very loose without a noticeable change on the estimated space requirements, because space complexity increases only logarithmically with $u(N,S)$.

Complexity. We use N to denote the length of the sliding window, either in number of arrivals or in time (depending on the desired sliding window model), and $u(N,S)$ as defined earlier. Also, $g(N,S) = \max(u(N,S), N)$; function g is used to enable unified cost expressions for both the time-based and count-based sliding window model.

To get an ϵ_{sw}-approximation of the number of one-bits in the sliding window, exponential histograms require $O(\ln N + \ln\ln(u(N,S)))$ memory per bucket, to store the bucket size and bucket boundaries. The number of buckets is $O(\ln(u(N,S))/\epsilon_{sw})$, yielding a total memory of $O(\ln^2(g(N,S))/\epsilon_{sw})$. The update cost per element is $O(\ln(u(N,S)))$ worst-case, and $O(1)$ amortized time. Queries covering the whole sliding window are executed in constant time. For queries with range $N' < N$, the required time is $O(\ln(u(N,S)/\epsilon_{sw}))$. The extra time is required for finding the oldest bucket overlapping with the query, assuming sequential access. If the storage model of the buckets supports random access, e.g., a fixed-length array, then this time can be further reduced to $O(\ln(\ln(u(N,S)/\epsilon_{sw})))$ with binary search.

The space complexity of ECM-sketches is as follows. For the Count-Min array, we require an array of width $w = \lceil e/\epsilon_{cm} \rceil$ and depth $d = \lceil \ln(1/\delta) \rceil$. Each cell in the array stores an exponential histogram, requiring $O(\ln^2(g(N,S))/\epsilon_{sw})$ bits. Therefore, the total required memory is $O(\frac{1}{\epsilon_{sw}\epsilon_{cm}} \ln^2(g(N,S)) \ln(1/\delta)) =$

$O(\frac{1}{\epsilon^2}\ln^2(g(N,S))\ln(1/\delta))$. Concerning time complexity, adding an element requires computing d hash functions, and updating d separate exponential histograms. The amortized complexity for each arrival is therefore $O(d) = O(\ln(1/\delta))$, whereas the worst-case complexity is $O(d\ln(u(N,S))) = O(\ln(u(N,S))\ln(1/\delta))$. Finally, query execution takes $O(\ln(1/\delta))$ time for a query of range N' equal to N. For $N' < N$, the execution cost is $O(d\ln(u(N,S))/\epsilon_{sw}) \leq O(\ln(1/\delta)\ln(u(N,S))/\epsilon)$ with sequential access to buckets, e.g., using a linked list. With random access support, binary search can be used for finding the last relevant bucket for each query, reducing the query cost to $O(\ln(1/\delta)\ln(\ln(u(N,S))/\epsilon))$.

4.2 Alternative Algorithms for Sliding Windows

Sliding window counters can also be materialized using other sliding window algorithms. In the literature, two such algorithms are particularly well-known: (a) deterministic waves, and, (b) randomized waves [21]. We now show how ECM-sketches can incorporate these algorithms, and discuss the positive and negative aspects of each variant.

Deterministic Waves. Deterministic waves [21] have identical memory requirements with exponential histograms, and they outperform exponential histograms with respect to worst-case complexity for updates, requiring always constant time. As such, the space and computational complexity of ECM-sketches based on deterministic waves is identical to that of sketches based on exponential histograms, with the only difference being the worst-case update complexity, which is $O(\ln(1/\delta))$.

A downside of deterministic waves is that they require knowledge of the upper bound of the number of arrivals $u(N,S)$ during the initialization of the data structures, to decide on the required number of queues/levels. Any overestimation of $u(N,S)$ is therefore translated to increased space requirements – logarithmic with $u(N,S)$. It is important to note that this constraint is substantially less limiting compared to the constraints of previous algorithms, e.g., [35], which required an upper bound for the total number of items in *all* streams, and therefore could not be applied to dynamic networks with an unknown number of participating nodes and streams.

Randomized Waves. Randomized waves [21] provide (ϵ,δ) approximation for the basic counting problem, i.e., $Pr[|\hat{x} - x| \leq \epsilon_{sw}x] \geq 1-\delta_{sw}$, where $\hat{x}$ and x denote the estimated and real number of true bits in the sliding window range respectively. They have substantially higher space complexity compared to their deterministic counterparts – $O(\ln(1/\delta_{sw})/\epsilon_{sw}^2)$ instead of $O(1/\epsilon_{sw})$. Nevertheless, they are important for distributed applications as they enable composition without causing an inflation of the worst-case error bounds; deterministic counterparts did not originally support any composition functionality. Therefore, we also consider randomized waves for integration with our ECM-sketch structures.

Theorem 3 *For any ϵ within $(0,1)$, an ECM-sketch constructed with $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$, $\epsilon_{sw} = \epsilon$, and $\delta_{sw} = \delta_{cm} = \delta/2$ satisfies $Pr[|\hat{f}(x,r) - f(x,r)| \leq \epsilon||a_r||_1] \geq 1-\delta_{sw}-\delta_{cm}$. Furthermore, the aforementioned combination of ϵ_{cm} and ϵ_{sw} minimizes the space complexity of the sketch.*

Proof In the appendix.

The space complexity of ECM-sketches based on randomized waves is derived by multiplying the space complexity of the two basic structures: $O\left(\ln(1/\delta_{cm})\ln(1/\delta_{sw})\ln^2(g(N,S))/(\epsilon_{cm}\epsilon_{sw}^2)\right) = O\left(\ln^2(\delta)\ln^2(g(N,S))/\epsilon^3\right)$. Inserting a new element requires $O(\ln(\delta_{cm})\ln(\delta_{sw})) = O(\ln^2(\delta))$ amortized time, and $O(\ln(\delta_{cm})\ln(\delta_{sw})\ln(u(N,S))) = O(\ln^2(\delta)\ln(u(N,S)))$ worst-case time. Finally, query execution takes $O(\ln(\delta_{cm})\ln(\delta_{sw})\ (\ln(u(N,S)) + 1/\epsilon_{sw}^2)) = O(\ln^2(\delta)(\ln(u(N,S)) + 1/\epsilon^2))$ with sequential access to buckets and $O(\ln(\delta_{cm})\ln(\delta_{sw})\ (\ln\ln(u(N,S))+\ln(1/\epsilon_{sw}^2))) = O(\ln^2(\delta)(\ln\ln(u(N,S))+\ln(1/\epsilon_{sw}^2)))$ time with random access.

Table 2 summarizes the main results for the combination of ECM-sketches and the three sliding window structures. The results correspond to both time-based and count-based sliding windows.

5 Order-Preserving Merging

For many distributed applications, such as the network monitoring application described in the introduction, we require merging of individual ECM-sketches $CM_1, CM_2, \ldots, CM_n$, each one corresponding to stream S_1, S_2, …, S_n, to get a single ECM-sketch $CM_\oplus$ that corresponds to the logical stream $S_\oplus = S_1 \oplus S_2 \oplus \ldots \oplus S_n$. The $\oplus$ operator is defined as a merging operator that preserves the ordering and arrival time of the events. Standard Count-Min sketches allow merging, as long as all sketches are constructed with identical dimensions and hash functions. For this, they rely on the linearity of the Count-Min counters, which are simple integers in the general case. However, this does not trivially hold for ECM-sketches, where the counters are not simple numbers but complex sliding window structures, since exponential histograms (as well as all other deterministic sliding window structures), do not support this kind of merging. Although randomized structures enable lossless merging (cf. Section 5.2), they come with a substantially higher space complexity, and are thus not preferable for ECM-sketches. Therefore, we first consider the order-preserving merging of

	Exponential Histogram	Deterministic Wave	Randomized Wave
Memory	$O\left(\frac{1}{\epsilon^2}\ln(\frac{1}{\delta})\ln^2(g(N,S))\right)$	$O\left(\frac{1}{\epsilon^2}\ln(\frac{1}{\delta})\ln^2(g(N,S))\right)$	$O\left(\frac{1}{\epsilon^3}\ln^2(\delta)\ln^2(g(N,S))\right)$
Amort. update	$O(\ln(1/\delta))$	$O(\ln(1/\delta))$	$O(\ln^2(\delta))$
Worst update	$O(\ln(1/\delta)\ln(u(N,S)))$	$O(\ln(1/\delta))$	$O(\ln^2(\delta)\ln(u(N,S)))$
Query	$O(\ln(1/\delta)\ln(u(N,S))/\epsilon)$	$O(\ln(1/\delta)\ln(u(N,S))/\epsilon)$	$O(\ln^2(\delta)(\ln(u(N,S))+1/\epsilon^2))$

Table 2 Computational and space complexity of ECM-sketches. Function $g(N,S)$ is used as a shortcut for $\max(u(N,S),N)$.

deterministic sliding window structures. Note that this problem is interesting in itself, since these data structures are widely used in the literature for maintaining statistics over sliding windows. We then extend our results to cover merging of randomized waves, and of ECM-sketches.

For completeness, before presenting the details of our merging algrorithm, we note that other types of merging are also possible. For example, Gibbons and Tirthapura [21], have considered utilizing more than one randomized waves for generating their position-wise union, i.e., for maintaining count-based sliding window statistics. Their scenario and query types are fundamentally different than ours.

5.1 Merging of Exponential Histograms

Consider a set of exponential histograms $EH_1, EH_2, \ldots, EH_n$, summarizing time-based sliding windows. All are configured to cover a sliding window of N time units. The merging operation is denoted with $\oplus$, i.e., $EH_\oplus = EH_1 \oplus EH_2 \oplus \ldots \oplus EH_n$. With EH_i^j we denote bucket j of EH_i, and $|EH_i^j|$ denotes the bucket size (number of true bits). By convention, buckets are numbered such that bucket 1 is the most recent. The ending time of the bucket is denoted as $e(EH_i^j)$. To ease exposition, we use $s(EH_i^j)$ to denote the starting time of the bucket, even though this is not explicitly stored in the buckets. By construction, the starting time of a bucket is equal to the ending time of the previous bucket, i.e., $s(EH_i^j) = e(EH_i^{j-1})$.

To construct $EH_\oplus$ our methodology considers the individual exponential histograms as logs. The basic idea is to reconstruct $EH_\oplus$ by assuming that half of the elements arrive at the starting time of each bucket, and the remaining at the ending time of the bucket. Precisely, let $\mathcal{B}$ denote the list containing all buckets of all sliding windows. We initialize an empty time-based exponential histogram with error ϵ', configured to keep the last N time units, and a maximum of $\sum_{i=1}^{n}|EH_i|$ elements. For each bucket $\mathcal{B}[i] \in \mathcal{B}$, we simulate the insertion in $EH_\oplus$ of $|\mathcal{B}[i]|$ true bits. Half of the bits are inserted with timestamp $s(\mathcal{B}[i])$, and the other half at time $e(\mathcal{B}[i])$. Insertions are simulated in the order defined by the starting and ending timestamps of the buckets.

Theorem 4 *Consider n time-based exponential histograms EH_1, EH_2, ..., EH_n, initialized with error parameter ϵ, and covering the same time range. The exponential histogram $EH_\oplus$ initialized with error parameter ϵ', and constructed with the proposed merging algorithm answers any query within its time range for the stream $S_\oplus$ with a maximum relative error of $(\epsilon + \epsilon' + \epsilon\epsilon')$.*

We will now give the intuition of the proof. The formal proof is presented in the appendix. Each exponential histogram EH of stream S configured with error parameter ϵ can be used to reconstruct an approximate stream S', as follows: For each bucket b in EH, add $|b|/2$ true bits in time $s(b)$, and $|b|/2$ true bits in time $e(b)$. We argue that answering any query with starting time s_q within the range of EH using the reconstructed stream S' will result to a maximum relative error ϵ. Let b_j be the bucket s.t. $s(b_j) < s_q \leq e(b_j)$. Therefore, the accurate answer x of the query for stream S is lower bounded by $l = \sum_{i=1}^{j-1}|b_i| + 1$ and upper bounded by $h = \sum_{i=1}^{j-1}|b_i| + |b_j|$. By construction, the reconstructed stream will contain a total of $\sum_{i=1}^{j-1}|b_i| + |b_j|/2$ items with timestamp greater than or equal to s_q. Therefore, answering the query by counting the number of true bits in the reconstructed stream with timestamp after s_q will have a maximum error of $\max(h - \sum_{i=0}^{j-1}|b_i| + |b_j|/2, \sum_{i=0}^{j-1}|b_i| + |b_j|/2 - l) = |b_j|/2$. By invariant 1 of exponential histograms, $|b_j|/2 \leq \epsilon(1 + \sum_{i=1}^{j-1}|b_i|) \leq \epsilon x$. Therefore, the maximum difference between the answer estimated by stream S' and the correct answer x will be less than or equal to ϵx.

Our merging algorithm is equivalent to reconstructing each stream S_i' from exponential histogram EH_i, and using these to recreate an exponential histogram $EH_\oplus$. The reconstruction of stream S' introduces a maximum relative error ϵ, as explained above. Summarizing S' with a new exponential histogram we get an additional error ϵ'. However, ϵ' is relative on the answer provided by stream S', and not by S. Therefore, the absolute error due to the exponential histogram summarization will be $\epsilon' x'$, where $x' \in (1 \pm \epsilon)x$ and x denoting the accurate answer on S_i. Summing both errors, we get a total relative error of $\epsilon + \epsilon' + \epsilon\epsilon'$.

For the special case when $\epsilon' = \epsilon$, the maximum relative error becomes $2\epsilon + \epsilon^2$. Concerning space and computational complexity, $EH_\oplus$ behaves as a standard exponential

histogram, and therefore has the same complexity as presented in [16]. □

Multi-level Merging. It is frequently desired to merge sliding windows in more than one levels. For example, consider a hierarchical P2P network, where each peer maintains its own exponential histogram, and pushes it to its parent for merging at regular intervals. Since the merged exponential histograms have the same properties as the individual exponential histograms (albeit with a higher ϵ), the above analysis also supports iterative merging of exponential histograms.

There are two types of approximation error that influence the estimation of a merged exponential histogram. A possible approximation error, denoted as err_1, is introduced due to halving of the size of the last bucket of the merged exponential histogram. This error occurs only at query time, and is independent of the number of performed merges. Therefore, at a multi-level merging scenario this error does not need to be propagated at the intermediary exponential histograms. A second type of error, termed as err_2, occurs due to the inclusion (exclusion) of data that arrived before (after) the query starting time in buckets that are accounted (not accounted) in the query result.

It turns out that the error err_2 is additive at the worst case (in absolute value). For instance, in the lowest level (Level 0) of the hierarchy, merging two exponential histograms (all with relative error ϵ), having a true number of bits (in a given query range) equal to i_1 and i_2, will result at a maximum value for $\text{err}_2 \leq \epsilon(i_1 + i_2)$. In Level 1, in addition to the previous possible errors, $\epsilon(i_1+i_2)+\epsilon(i_3+i_4)$ stream items may be incorrectly registered at the wrong side of the query start time. A recursive repetition for h levels results to $\text{err}_2 \leq h\epsilon i$, where $i = \sum_j i_j$. The total absolute error (including err_1) then becomes $\text{err} = \text{err}_2 + \text{err}_1 \leq h\epsilon i + \epsilon(i + h\epsilon i)$, resulting to a maximum relative error of $h\epsilon(1 + \epsilon) + \epsilon$.

In many applications, the number of merging levels can be predicted, or even controlled when constructing the network topology. For example, consider DHT-based or hierarchical P2P topologies, which typically enable a balanced-tree access to the peers of height $h = \log(N)$, where N is the number of nodes. In such systems, initializing the individual exponential histograms with error $\frac{\sqrt{1+2h+h^2+4h\epsilon}-1-h}{2h}$ yields a final merged exponential histogram of relative error ϵ. Naturally, this causes a slight inflation of the size of the sliding window, by $O(\log(N))$. However, even with this inflation, exponential histograms are – even for extremely large networks – substantially smaller and more efficient than randomized data structures that enable error-free merging in the expense of memory proportional to $O(\ln(1/\delta)/\epsilon^2)$ (see also Section 5.2).

Deterministic Waves. The merging technique trivially extends for deterministic waves. Recall that each wave is composed of l levels, each covering a different range. To perform the merging, we start from the lowest wave level $l - 1$, and switch to a higher level every $(1/\epsilon + 1)/2$ bits, i.e., when the first entry in the higher level has arrived before the next entry in the current level. Repeating the calculation of the error bounds for the merging of deterministic waves becomes straightforward when we notice that invariant 1 of the exponential histograms is also true for deterministic waves.

	EH_1		EH_2				
Bucket id	2	1	5	4	3	2	1
Size	1	1	8	4	2	1	1
Completion time	3	20	3	5	10	15	19
Arrivals	500	1000	900	950	980	990	1000

Fig. 2 An example why merging of count-based exponential histograms is not possible.

Count-based Exponential Histograms. Although exponential histograms cover both time-based and count-based sliding windows, merging of exponential histograms is specific to time-based sliding windows. Count-based sliding windows do not contain sufficient information for enabling order-preserving merging. Even storing the system-wide time of the buckets would not be sufficient to allow such a merging. To illustrate this limitation, consider the two count-based exponential histograms depicted in Fig. 2. For each bucket we store the size of the bucket, the bucket completion time and the total number of arrivals until that time. An arrival in count-based sliding windows might be a true or a false bit. An example query can then be: *how many true bits arrived in the last 100 system-wide arrivals*. If these 100 system-wide arrivals were read between time 19 and 20, then the correct answer would be 1. However, it is also possible that the last 100 system-wide arrivals have arrived between time 3 and time 20, in which case the correct answer could be anything between 2 and 9. The information contained in the two exponential histograms is not sufficient to estimate this type of queries, as it only allows us to preserve the order of the true bits, but looses the order of the false bits, which is also important. Therefore, given only the exponential histograms, it is not possible to merge them in a way that preserves the ordering of both true and false bits. Deterministic and randomized waves also have the same limitation when it comes to order-preserving merging of count-based sliding windows.

5.2 Merging of Randomized Waves

Randomized waves were proposed in [21] to address the problem of distributed union counting: *counting the number of 1's in the position-wise union of t distributed data streams, over a sliding window*. Even though the algorithm of [21] can utilize more than one waves constructed at different nodes to answer queries, it does not consider merging of several waves to generate a single wave. Instead, it as-

sumes that individual randomized waves can be stored and accessed any time, which is inconvenient for large networks. To eliminate this assumption, we now describe a slight variation of the initial algorithm that can produce a single randomized wave out of a set of individual waves, with the same probabilistic accuracy guarantees as the individual waves.

Our algorithm simulates the construction of the merged randomized wave $RW_{\oplus}$ by using only the information included in the individual randomized waves. Consider a set $\mathcal{R}$ of randomized waves RW_1, RW_2, ..., RW_n, configured to store a sliding window of N time units, with error parameters ϵ and δ. The merged randomized wave $RW_{\oplus}$ is initialized with the same ϵ and δ parameters, for storing a maximum of $\sum_{i=1}^{n} |RW_i|$ events over N time units. Each level l of $RW_{\oplus}$ is then constructed by concatenating the corresponding level l from all individual randomized waves, sorting all events based on the timestamp, and keeping the last c/ϵ^2 events. Recall that the number of levels of individual randomized waves is determined based on the maximum number of events in the sliding window. Therefore, it may happen that $RW_{\oplus}$ has more levels than the individual randomized waves. To populate the lower levels of $RW_{\oplus}$, we rehash the events populating the last level of each individual randomized wave, as proposed in [21] when merging different levels from randomized waves.

The process of query execution and the accuracy guarantees remain the same as for the standard randomized waves.

5.3 Merging of ECM-Sketches

Consider a set of ECM-sketches CM_1, CM_2, ..., CM_n with identical dimensions and hash functions. The ECM-sketch $CM_{\oplus}$ with each counter set to the sum of all corresponding counters from the individual sketches (as defined by the $\oplus$ operator), summarizes the information found in the individual sketches:

$$CM_{\oplus}[j,k] = CM_1[j,k] \oplus CM_2[j,k] \oplus \ldots \oplus CM_n[j,k]$$

To bound the estimation error, we consider the two sources of error in the merged ECM-sketch. The error due to the Count-Min sketch ϵ_{cm} does not change, since it only depends on the dimensionality of the Count-Min array, which is fixed. However, the error due to sliding window estimations at each counter might change with each merging. Let ϵ'_{sw} denote the error produced by the merging of the corresponding Count-Min counters, as discussed in Sections 5.1 and 5.2. If ϵ_{sw} and ϵ_{cm} are configured according to Theorem 3, it can be easily shown that ϵ'_{sw} will always be greater than or equal to $\epsilon_{cm}/(1-\epsilon_{cm})$. Then, the error bounds follow directly by Lemma 3: $|\hat{f}(x,r) - f(x,r)| \leq \epsilon'_{sw}||a_r||_1$ with probability $1-\delta_{cm}$.

6 Continuous Function Monitoring with ECM-Sketches

A substantial number of distributed applications requires continuous monitoring of complex functions defined over high-dimensional domains. For example, network administrators frequently require to monitor the (sliding-window) heavy-hitter IP addresses over distributed streams of network packets (e.g., received by the edge routers of the corporate network), as these IPs are potentially launching a DoS attack. ECM-sketches can be exploited in these applications, such that each network node can compactly and efficiently maintain its local state, as well as effectively propagate it over the network. In this section, we show how ECM-sketches can leverage the geometric method [33,27], to enable continuous function monitoring.

We illustrate our technique by addressing two frequent requirements of distributed applications: (a) monitoring items with frequency over a user-defined threshold τ, and, (b) monitoring self-join size queries. In principle, any query type that can be answered by (a sequence of) point queries can be monitored in the lines of the algorithm that we will present for query (a). Some examples include hierarchical heavy hitters, quantiles, range queries, and maximum frequency queries (see also [11,31] for a more detailed discussion on how centralized Count-min sketches and ECM-sketches can address these problems using point queries). It is also straightforward to extend the algorithm for query (b) for inner-product size queries.

Section 6.1 provides an introduction to the geometric method. In Section 6.2 we introduce the integration of ECM-sketches with the geometric method, and discuss the main challenges that need to be addressed. Then, in Section 6.3, we briefly discuss an algorithm for query (a). This discussion serves mainly as a first, simple, example for the integration. An algorithm for query (b) is presented in more detail in Sections 6.4 and 6.5. Our discussion for query (b) includes an efficient monitoring algorithm and novel theoretical results to enable dimensionality reduction of the monitoring problem (from $d \times w$ to d), which translates to drastic network savings and better scalability.

6.1 An Introduction to the Geometric Method

Sharfman et al. [33] consider the basic problem of monitoring *distributed threshold-crossing queries*; that is, monitoring whether $f(\mathbf{v}) < \tau$ or $f(\mathbf{v}) > \tau$ for a possibly complex, non-linear function f and a high-dimensional vector $\mathbf{v}$ computed as the aggregate of the corresponding local/partial vectors $\{\mathbf{v}(p_1), \mathbf{v}(p_2), \ldots, \mathbf{v}(p_n)\}$ at a set of n sites. The key idea of the method is, since it is generally impossible to connect the values of f on the local statistics vectors to the global value $f(\mathbf{v})$, one can employ geometric arguments

to monitor the *domain* (rather than the range) of the monitored function f. The monitoring protocol works as follows. Assume that at any point in time, each site p_i has informed the coordinator of some prior state of its local vector $\mathbf{v'}(p_i)$; thus, the coordinator has an estimated global vector $\mathbf{e} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{v'}(p_i)$. Clearly, the updates arriving at sites can cause the local vectors $\mathbf{v}(p_i)$ to drift too far from their previously reported values $\mathbf{v'}(p_i)$, possibly leading to a violation of the threshold τ. Let $\Delta\mathbf{v}(p_i) = \mathbf{v}(p_i) - \mathbf{v'}(p_i)$ denote the local *delta vector* (due to updates) at site i, and let $\mathbf{u}(p_i) = \mathbf{e} + \Delta\mathbf{v}(p_i)$ be the *drift vector* from the previously reported estimate at site p_i. We can then express the current global statistics vector $\mathbf{v}$ in terms of the drift vectors:

$$\mathbf{v} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{v'}(p_i)+\Delta\mathbf{v}(p_i)) = \mathbf{e}+\frac{1}{N}\sum_{i=1}^{N}\Delta\mathbf{v}(p_i) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{u}(p_i).$$

That is, the current global vector is a convex combination of drift vectors and, thus, guaranteed to lie somewhere within the convex hull of the delta vectors around $\mathbf{e}$. Fig. 3 depicts an example in $d = 2$ dimensions. The current value of the global statistics vector lies somewhere within the shaded convex-hull region; thus, as long as the convex hull does not overlap the inadmissible region (i.e., the region $\{\mathbf{v} \in \mathbb{R}^2 : f(\mathbf{v}) > \tau\}$ in Fig. 3) we can guarantee that the threshold has not been violated (i.e., $f(\mathbf{v}) \leq \tau$).

The problem, of course, is that the $\Delta\mathbf{v}(p_i)$'s are spread across the sites and, thus, the above condition cannot be checked locally. To transform the global condition into a local constraint, we place a d-dimensional *bounding ball* $B(\mathbf{e}, \Delta\mathbf{v}(p_i))$ around each local delta vector, of radius $\frac{1}{2}\|\Delta\mathbf{v}(p_i)\|$ and centered at $\mathbf{e} + \frac{1}{2}\Delta\mathbf{v}(p_i)$ (see Fig. 3). It can be shown that the union of these balls completely covers the convex hull of the drift vectors [33]. This observation effectively reduces the problem of monitoring the global statistics vector to the local problem of each remote site monitoring the ball around its local delta vector.

More specifically, given the monitored function f and threshold τ, we can partition the d-dimensional space to two regions $V = \{\mathbf{v} : f(\mathbf{v}) > \tau\}$ and $\overline{V} = \{\mathbf{v} : f(\mathbf{v}) \leq \tau\}$. (Note that each of these can be arbitrarily complex, e.g., they may comprise multiple disjoint regions of $\mathbb{R}^d$.) The basic protocol is now quite simple: Each site monitors its delta vector $\Delta\mathbf{v}(p_i)$ and, with each update, checks whether its bounding ball $B(\mathbf{e}, \Delta\mathbf{v}(p_i))$ is *monochromatic*, i.e., all points in the ball lie within the same region (either V, or $\overline{V}$). If this is not the case, we have a *local threshold violation*, and the site communicates its local $\Delta\mathbf{v}(p_i)$ to the coordinator. The coordinator then initiates a *synchronization process* that typically tries to resolve the local violation by communicating with only a subset of the sites in order to "balance out" the violating $\Delta\mathbf{v}(p_i)$ and ensure the monochromicity of all local bounding balls [33]. Briefly, this process involves collecting the current delta vectors from (a subset of) the sites, and recomputing the minimum and maximum values of $f(\mathbf{v})$ according to the new, partial, average. If both values reside at the same side of the threshold, the coordinator computes a slack vector for each site in the synchronization set that shifts the local vector to the partial average. In the worst case, the delta vectors from all N sites are collected, leading to an accurate estimate of the current global statistics vector, which is by definition monochromatic (since all bounding balls have 0 radius).

In more recent work, Sharfman et al. [27] show that the local bounding balls defined by the geometric method are actually special cases of a more general theory of *Safe Zones (SZs)*, which can be broadly defined as *convex subsets of the admissible region* of a threshold-crossing query. Then, as long as the local drift vectors stay within such a SZ, the global vector is guaranteed (by convexity) to be within the admissible region of the query.

6.2 ECM-Sketches and the Geometric Method

We are interested in domains where the local and global statistics vectors ($\mathbf{v}(p_i)$ and $\mathbf{v}$ respectively) are defined over a user-chosen sliding window range, and are expected to be high-dimensional, e.g., they may contain the frequency of each item within the user-defined sliding window, for a large number of items. Clearly, accurate maintenance of these statistics for high-velocity data streams is computationally challenging. Furthermore, the aggregation of the local statistics vectors in order to compute the global statistics vector is costly, since it requires exchanging large vectors during synchronization. Both computational and network cost can be substantially reduced with a small trade-off on quality, by using ECM-sketches. This requires the following modifications in the geometric method: a) sites use ECM-sketches to approximate their local statistics vectors, b) the global statistics/estimate vector, the local delta vectors and the drift vectors, are all represented as Count-min sketches, extracted by the ECM-sketches (at query time), and finally, c) during configuration of the geometric method, the query is described on top of the sketch representations of the local and global statistics vector.

Clearly, a naive implementation of the above changes would be subject to substantial constraints, since the size of the domain space of geometric monitoring would be equal to the dimensionality of the ECM-sketches ($d \times w$), and the geometric method is known to be inefficient in high-dimensional domains. It is therefore imperative to reduce the dimensionality of the problems to monitor. In the following sections we show how this is achieved for the frequent items query and for the self-join size queries.

Before going into further details, notice that the above method enables concurrent monitoring of multiple queries (not necessarily of the same type) with a single ECM-sketch per node, which satisfies the strictest accuracy requirements

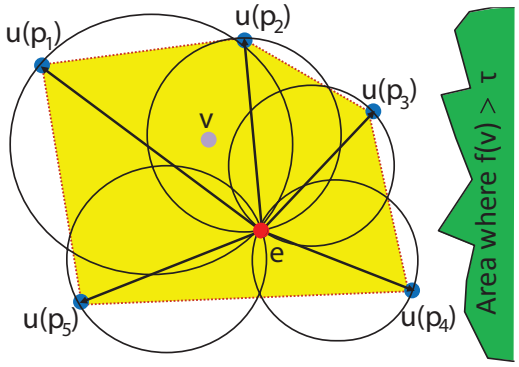

Fig. 3 Estimate vector $\mathbf{e}$, $\Delta\mathbf{v}(p_i)$ (arrows out of $\mathbf{e}$), drift vectors $\mathbf{u}$, convex hull enclosing the current global vector $\mathbf{v}$ (dotted outline), and bounding balls $B(\mathbf{e}, \Delta\mathbf{v}(p_i))$.

of all queries and covers the largest window. Multiple instances of the geometric method (in the simple case, one per query), could then be executed in parallel, coordinating the synchronization process to reduce network cost. The network cost for monitoring the queries is determined by the network requirements of the geometric method, i.e., it depends mainly on the stability of the answer and the acceptable error parameter θ. Fully analyzing all query types, examining the involved challenges, and exploiting the parallel execution of the queries for network and computational benefits is an interesting open problem, and part of our future work.

6.3 Monitoring Frequent Items

Let $p_1, p_2, \ldots, p_n$ denote all n network nodes, $S_1, S_2, \ldots, S_n$ their corresponding streams, and S_0 the order-preserving union of these streams. We use $\mathcal{D}$ to denote the domain of S_0, i.e., all distinct items appearing in the stream, and $S_{i,r}$ the sub-stream of S_i within query range r. The algorithm addresses the problem of distributed continuous monitoring of the set $\mathcal{F}_\tau^r$ of items with frequency in $S_{0,r}$ greater than a user-chosen threshold τ. It works by decomposing the problem to a set of smaller individual problems, one for each distinct item occurring in the stream, yet without requiring the knowledge of all distinct items a priori.

The user first selects the frequency threshold τ, the desired accuracy of ECM-sketches (δ and ϵ), and an acceptable error parameter $\theta \geq 0$ that defines the error tolerance of the geometric method algorithm, i.e., it is acceptable for the algorithm to misclassify items with frequency in the range of $\tau(1 \pm \theta)$. At initialization time t_0, each site p_i constructs an empty ECM-sketch ECM_i to be used as its local statistics vector, and an empty Count-min sketch $CM(t_0)$ to be used as the reference vector (we drop the site id from the notation since the reference vector is always identical at all sites). Both sketches are of the same size $d \times w$, and are initialized with identical hash functions at all sites. After initialization, sites enter the monitoring phase: For each item $x \in \mathcal{D}$, we define a d-dimensional threshold-crossing query as the boolean condition:

$$Q(\boldsymbol{f}, \mathbf{v}, x, \tau, \theta) \equiv \begin{cases} \boldsymbol{f}(\mathbf{v}(t,x)) \geq \tau(1-\theta) \text{ if } \boldsymbol{f}(\mathbf{v}(t_0,x)) < \tau \\ \boldsymbol{f}(\mathbf{v}(t,x)) < \tau(1+\theta) \text{ if } \boldsymbol{f}(\mathbf{v}(t_0,x)) \geq \tau \end{cases}$$

with function vector $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}$ defined as $\boldsymbol{f}(\mathbf{v}) = n \times \min_{j=1}^{d} \mathbf{v}[j]$. The d-dimensional vectors $\mathbf{v}(t,x)$ and $\mathbf{v}(t_0,x)$ are extracted by ECM_i and CM respectively, as follows. $\mathbf{v}(t)[j] = E_i(j, h_j(x), r)$ (the estimation from the counter of ECM_i at position $(j, h_j(x))$) and $\mathbf{v}(t_0)[j] = CM[j, h_j(x)]$ (the value of the counter of the reference Count-min sketch at position $(j, h_j(x))$).

Using the geometric method, sites monitor the threshold-crossing queries in order to detect item arrivals or expirations that potentially invalidate the set of estimated frequent items $\hat{\mathcal{F}}_\tau^r$. An arrival of any item x is handled as follows. First, the local ECM-sketch is updated to include the arrival. If x is already frequent, nothing else needs to be done. In the opposite case, the site probes the corresponding threshold query $Q(\boldsymbol{f}, \mathbf{v}, x, \tau, \theta)$, initiating a synchronization if threshold crossing occurs. Notice that, for synchronization, the coordinator needs to collect only the values of the ECM-sketch counters corresponding to x, i.e., $E(j, h_j(x), r)$ for $j = 1 \ldots d$, in order to update the reference Count-min sketch and decide whether the item causing the violation is frequent. The actual sliding window structures do not need to be exchanged.

Counter updates due to expirations are slightly more complicated (these could cause the removal of a frequent item from $\hat{\mathcal{F}}_\tau^r$). The technical challenge comes from the fact that a bucket expiration at the sliding window of any counter from the ECM-sketch may affect many items, introducing computational complexity. One approach would be to have each site execute the threshold crossing queries for all frequent items at regular intervals. To reduce computational complexity, each site p_i instead maintains a balanced binary search tree that contains all counters of ECM_i and the set of frequent items corresponding to each counter, ordered by the expiration time of their oldest bucket. This tree enables p_i to quickly detect (in constant time) whether any of the counters of ECM_i contains expired buckets, and test only the relevant threshold-crossing queries. The quality guarantees and memory footprint of the above algorithm are summarized by the following lemma.

Lemma 1 *The algorithm guarantees that with probability greater than or equal to* $1 - \delta$*, any item* x *contained in* $\hat{\mathcal{F}}_\tau^r$ *has a real frequency in* $S_{0,r}$ *greater than* $(1-\theta)\tau - \epsilon||S_{0,r}||_1$*, whereas any item* x *not contained in* $\hat{\mathcal{F}}_\tau^r$ *has a real frequency less than* $(1+\theta)\tau + \epsilon||S_{0,r}||_1$*. The algorithm requires memory of* $O(\frac{1}{\epsilon^2}\ln(\frac{1}{\delta})\ln^2(||r||_1) + |\hat{\mathcal{F}}_\tau^r|)$.

6.4 Monitoring Self-Join Size Queries

In the previous case, the problem to be monitored was always d-dimensional, with a small d ($d \leq 5$ for $\delta \geq 0.01$). As such, the geometric method was able to bound the convex hull using relatively small balls, effectively filtering out local updates. Furthermore, threshold violations could be resolved by exchanging only d counters. Estimation of the self-join size, however, involves all $d \times w$ counters, with $(d \times w)$ typically in the hundreds. A naive application of the geometric method for self-join size monitoring would therefore require exchanging $d \times w$ counters at each threshold violation. The problem is further aggravated by the high dimensionality of the bounding balls (equal to the number of counters), which increases the frequency of threshold crossings.

Our attack to this problem is twofold. First, we adapt a recently-proposed insight [18] that enables us to reduce the problem to d dimensions, by monitoring upper and lower bounds of the self-join size estimate instead. This adaptation includes repeating the analysis of [18] for the ECM-sketch (monitoring the min instead of the median, and providing error guarantees relevant to the stream length). However, the bounds offered by this method alone turn up to be quite loose when it comes to ECM-sketches, causing frequent threshold crossings. Therefore, we offer a new, second, approach that further tightens these bounds by exploiting the sliding-window property of ECM-sketches. Compared to the first approach, the second approach drastically reduces the number of threshold crossing, enabling substantial network gains.

We initiate the discussion with some basic notation. Let r denote the query range, and $\mathbf{v}_i(t)$ the Count-min sketch extracted by the ECM-sketch of node p_i, with each counter computed as $\mathbf{v}_i(t)[row, col] = E_i(row, col, r)$.[1] Also, $\mathbf{v}(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i(t)$ (the average of $\mathbf{v}_i(t)$). Function $\boldsymbol{f}(\mathbf{v})$ corresponds to the self-join size estimate function with Count-min sketches, i.e., $\boldsymbol{f}(\mathbf{v}) = \min_{row=1}^{d} \sum_{col=1}^{w} (\mathbf{v}[row, col])^2$. Finally, $\boldsymbol{d}_i$ is the d-dimensional vector computed as $\boldsymbol{d}_i[row] = ||\mathbf{v}_i(t)[row] - \mathbf{v}_i(t_0)[row]||$, and $\boldsymbol{d} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{d}_i$. The following lemma enables us to extract d-dimensional threshold-crossing queries:

Lemma 2 *If* $\min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} - \boldsymbol{d}[row]\} \geq \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1+\theta}}$ *and* $\min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} + \boldsymbol{d}[row]\} \leq \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}$, *then* $\boldsymbol{f}(\mathbf{v}(t_0)) \in (1 \pm \theta)\boldsymbol{f}(\mathbf{v}(t))$.

Proof In the appendix.

[1] The geometric method is trivially extended to handle matrices instead of vectors by applying vectorization on the matrices, and adjusting the monitored function to use the corresponding vector dimensions. We use the matrix notation for the sketches only for convenience.

Since $\boldsymbol{d}$ is a convex combination (the average) of the distributed values $\boldsymbol{d}_j$, we can already exploit the geometric method to monitor the self-join size estimate. This can be achieved by defining two queries, the first (Q_u) for upper-bounding $\min_{i=1}^{d}\{||\mathbf{v}(t_0)[i]|| - n\boldsymbol{d}[i]\}$, and the second ($Q_l$) for lower-bounding $\min_{i=1}^{d}\{||\mathbf{v}(t_0)[i]|| + n\boldsymbol{d}[i]\}$. A key observation, however, is that the definition of $\boldsymbol{d}$ does not account for the direction of each update: any update of a counter on a local ECM-sketch that shifts a counter away from its last synchronized value (either decreasing the counter value due to an expiration of a bucket, or increasing the value due to a new arrival) will lead to an increase of $\boldsymbol{d}$. This, however, results in unnecessary threshold crossings. For example, an increase of a counter at a peer p_j may lead to a threshold crossing on the lower-bound threshold query, even though in practice an increase of the counter can only lead to an increase of the self-join size.

To circumvent this problem we introduce two auxiliary matrices per peer, one for the upper bound that includes only the counter shifts which increase the counters' values since last synchronization, and another with the shifts that decrease the counters' values. Formally, for the upper bound, $\mathbf{v}_i^u(t)$ is computed as: $\mathbf{v}_i^u(t)[row, col] = \max(\mathbf{v}_i(t), \mathbf{v}_i(t_0))$, and for the lower bound $\mathbf{v}_i^l(t)[row, col] = \min(\mathbf{v}_i(t), \mathbf{v}_i(t_0))$. Similarly, $\boldsymbol{d}_i^u = ||\mathbf{v}_i^u(t)[row] - \mathbf{v}(t_0)[row]||$ and $\boldsymbol{d}_i^l = ||\mathbf{v}_i^l(t)[row] - \mathbf{v}(t_0)[row]||$. $\boldsymbol{d}^u$ and $\boldsymbol{d}^l$ are the corresponding averages over all nodes. Then, we can show the following:

Theorem 5 *If* $\min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} - \boldsymbol{d}^l[row]\} \geq \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1+\theta}}$ *and* $\min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} + \boldsymbol{d}^u[row]\} \leq \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}$, *then* $\boldsymbol{f}(\mathbf{v}(t_0)) \in (1 \pm \theta)\boldsymbol{f}(\mathbf{v}(t))$.

Proof In the appendix.

This leads to the following threshold crossing queries (the queries become true when threshold violation occurs):

$$Q_u(\boldsymbol{f}, \boldsymbol{d}^u, \mathbf{v}, \theta) \equiv \min_{row=1}^{d}\{\boldsymbol{d}^u[row] + \frac{||\mathbf{v}(t_0)[row]||}{n}\} > \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}$$

and for the lower bound:

$$Q_l(\boldsymbol{f}, \boldsymbol{d}^l, \mathbf{v}, \theta) \equiv \min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} - \boldsymbol{d}^l[row]\} < \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1+\theta}}$$

Synchronization. A two-phase synchronization algorithm is used to handle threshold violations. Without loss of generality we will demonstrate the algorithm assuming a threshold violation in Q_u (the case of Q_l is analogous). In a first phase, all nodes p_i send their local values of $\boldsymbol{d}_i^u$ to the coordinator in order to compute the accurate average value $\boldsymbol{d}^u$. If the updated $\boldsymbol{d}^u$ no longer causes threshold violation, it is sent back to all nodes to be used in the monitoring algorithm. However, if this first phase is not sufficient to address

the threshold violation, the coordinator collects also the local values of $\mathbf{v}_i(t)$, recomputes the average value $\mathbf{v}(t)$ and the updated self-join size, and reinitializes the monitoring algorithm with the updated values. Both phases can be further extended such that synchronization stops as soon as balancing between the retrieved vectors is possible, as explained in Section 6.1. The first phase has a network cost (in transfer volume) of $O(d \times n) = O(n \ln(1/\delta))$, whereas the cost of the (more infrequent) second phase is $O(d \times w \times n) = O(n \ln(1/\delta)/\epsilon)$.

6.5 Efficient Monitoring of the Minimum

The previous discussion has abstracted away the details of the geometric monitoring of functions containing the minimum. In principle, the standard geometric monitoring algorithm can be used, as described above. However, the nature of the monitored function enables substantial optimizations. We distinguish two types of queries: (a) the queries where the last estimate vector is located above the threshold, and (b) the queries where the last estimate vector is below the threshold.

For the first type of queries, we will use as a running example the query Q_l, introduced in Section 6.4 (the same principles apply to the queries introduced in Section 6.3 that correspond to frequent items). The admissible region of the monitored vector in this query is already convex (i.e., in two dimensions, this will be the L-shaped area above the threshold). Hence, the monochromicity test becomes fairly simple: a node p_i reading an update only needs to test whether the local value of $\boldsymbol{d}_i^l$ stays within the convex admissible region, in which case the update is guaranteed to be safe.

Query Q_u, and the queries from Section 6.3 corresponding to infrequent items belong to the second type of queries. For these queries, the admissible region is non-convex. However, the inadmissible region is now convex, and we can apply a different technique based on convex safe zones [27]: In the absense of statistics for the velocity and direction of $\boldsymbol{d}^u$, we choose the safe zone such that it maximizes the slack in all dimensions, as follows. First, we find the point p of the inadmissible region that is closest to $\frac{\mathbf{v}(t_0)}{n}$. It is easy to show that this point is $p[i] = \max\left(\frac{\mathbf{v}(t_0)}{n},\ \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}\right)$. Then, we find the hyperplane H passing from p that is perpendicular to vector $(p - \frac{\mathbf{v}(t_0)}{n})$ (see Fig. 4 for a two-dimensional illustration of this process). Hyperplane H divides the d-dimensional space to two convex subspaces. By construction, the one of the two subspaces (in the two-dimensional example, the subspace in the right of H, denoted with R) contains the inadmissible region and possibly some admissible area, whereas the second subspace (denoted with L) contains only admissible area w.r.t. query Q_u. Since L is convex, it can be used as a safe zone for the geometric method.

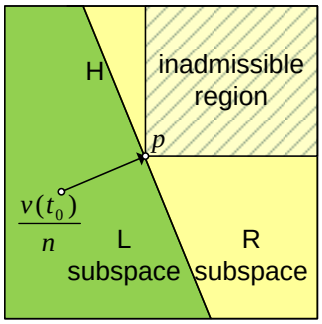

Fig. 4 Monitoring of the minimum for Q_u with safe-zones. The inadmissible region is fully covered by the R subspace (yellow). The L subspace (green) can be used as a safe zone for $\mathbf{v}(t_0)/n + \boldsymbol{d}^u$.

In particular, after each update, nodes only need to check whether $\mathbf{v}(t_0)/n + \boldsymbol{d}^u$ is still within L. This is guaranteed to be the case whenever $\mathbf{v}(t_0)/n + \boldsymbol{d}_i^u$ remains within L for all nodes i. The computational complexity for this process is only $O(d) = O(\ln(1/\delta))$ for computing the hyperplane and checking whether $\boldsymbol{d}_i^u$ is still in the safe zone.

7 Experimental Evaluation

Our experiments focused on evaluating ECM-sketches with respect to their scalability, effectiveness, and efficiency, as well as their suitability for distributed setups. The experiments were conducted using two large real-life data sets, the world-cup'98 [2] (*wc98*) and the CAIDA Anonymized Internet Traces 2011 data set (*caida*[2]). The wc98 data set consists of all HTTP requests that were directed within a period of 92 days to the web-servers hosting the official world-cup 1998 website. It contains a total of 1.089 billion valid requests, served by 33 server mirrors. Each request was indexed using the web-page url as a key, i.e., the ECM-sketch could be used for estimating the popularity of each web-page. The caida data set consists of Internet traces collected by passive monitors installed in Chicago and San Jose. For this experiment we have used the subset of data collected from the Chicago monitors in 17th February 2011, which contained a total of 345 Million IPv4 packets. Each packet was indexed using the source's IP address. Therefore, the ECM-sketch enabled estimating the number of packets sent by each IP address.

We compared three sketch variants, differentiating on the employed sliding window algorithm: (a) the default variant described earlier which is based on exponential histograms, denoted as *ECM-EH*, (b) a variant using deterministic waves (*ECM-DW*), and, (c) a variant based on randomized waves (*ECM-RW*). Comparison between the variants was performed to examine the influence of the sliding window algorithm to the performance of ECM-sketches, in both centralized

[2] Available from `http://www.caida.org/data/`

	ECM-EH	ECM-DW	ECM-RW
wc98			
Update rate	4343095	6067130	70468
Query rate	1641935	1850909	11905
caida			
Update rate	5344982	6205999	72778
Query rate	2377502	3827267	15108

Table 3 Indicative update and query rates (per second) for the centralized setup

ϵ	Data set	Point queries ECM-EH Centr.:Distr.	Inc.rate	Self join ECM-EH Centr.:Distr.	Inc.rate	ECM-RW Centr.:Distr.
0.1	wc98	0.018:0.020	1.114	0.018:0.019	1.053	0.017:0.017
0.2	wc98	0.034:0.040	1.181	0.034:0.037	1.092	0.031:0.031
0.1	caida	0.020:0.021	1.043	0.020:0.020	1.018	0.018:0.018
0.2	caida	0.038:0.041	1.079	0.037:0.039	1.042	0.034:0.034

Table 4 Observed errors and error increase rate – inflation is due to the iterative merging.

and distributed environments. Comparison with ECM-RW was of particular interest, since randomized algorithms for sliding window maintenance (such as the randomized waves employed by ECM-RW) were the only ones supporting merging prior to this work. Therefore, examining the performance of ECM-RW experimentally also serves the purpose of examining the importance of the merging mechanism for deterministic sliding window algorithms, proposed in this work.

7.1 Implementation Details

ECM-sketches were implemented in Java 1.7 using 32-bit addressing. The timing experiments were executed on a single idle core of an Intel Xeon E5-2450, clocked at 2.1 GHz. For the wc98 data set, deterministic and randomized waves were initialized with an upper bound of 1000 events per second, whereas for the caida dataset we have used an upper bound of 1000 events per millisecond. In practice, it is rarely possible to predict the maximum number of events per sliding window, and therefore such estimates (typically decided by analyzing a small stream sample) are often the only option. Exponential histograms did not require such knowledge at initialization time.

Particularly for randomized waves, Gibbons and Tirthapura [21,20] explain that a correctness probability of $1-\delta_{sw}$ requires the parallel maintenance of $c\ln_2(1/\delta_{sw})$ independent instances of the data structure, where the constant $c = 36$ is determined by worst-case analysis. This number of repetitions, in combination to the space complexity of each instance (c/ϵ_{sw}^2), can make the exchange of randomized waves over a distributed network extremely inefficient – hence the importance of the ability to merge deterministic data structures that is proposed in this article. Notice however that, as suggested in [20], smaller constants may also be used in practice in order to reduce the space and computational complexity. This, of course, comes at the cost of meaningless worst-case guarantees. In the following experiments, we set $c = \sqrt{36} = 6$, which reduces the cost by a factor of 36, but still offers an empirical estimation accuracy that is comparable to the one of deterministic sliding window structures configured with the same ϵ.

Unless otherwise mentioned, ECM-sketches were set to monitor a sliding window of 2 million time units. For wc98, this corresponded to 2 million seconds, i.e., 23 days, whereas for caida, it corresponded to 2 million microseconds, i.e., 33 minutes. Queries smaller than the sliding window were executed as well, using the same ECM-sketches. In particular, queries were generated with an exponentially increasing range, i.e., query q_i covered the range $[t-10^i, t]$, with t denoting the time of the last arrival. For each range, a self-join size query, as well as a set of point queries were constructed and executed. For thorough evaluation, we constructed one point query for each distinct item in the query range (i.e., estimating the popularity of each web-page in the wc98 dataset, or the number of packets sent by each IP address in the caida dataset).

7.2 Centralized Setup

In the centralized scenario, a single site monitors the whole stream and maintains an ECM-sketch, which is subsequently used for answering the queries. We first consider the trade-off between memory requirements and estimation error. For this, we vary ϵ within the range of $[0.05, 0.3]$, keeping $\delta = 0.15$. For each ϵ value, we use the analysis presented in Section 4 for point and self-join size queries to configure the ECM-sketch, such that the required memory for the targeted query type is minimized.

Figures 5(a)-(d) plot the average and maximum observed error in correlation to the required memory for the two data sets. The plots are annotated with indicative ϵ values. The displayed error at the Y axis is relative to the number of events arriving within the query range, i.e., for point queries, $\text{err} = |\hat{f}(x,r) - f(x,r)|/||a_r||_1$ and for self-joins, $\text{err} = |\widehat{a_r \odot a_r} - a_r \odot a_r|/(||a_r||_1)^2$. Recall that ECM-RW does not allow probabilistic guarantees for self-join size queries, and is therefore not considered for this type of queries. Table 3 presents indicative update and point query rates for the considered sketches.

We first observe that both the average and maximum observed errors are lower than the user-selected value ϵ for all ECM-sketch variants. However, the memory requirements of ECM-RW are typically two to three orders of magnitude higher than the requirements of ECM-sketches based on the two deterministic structures configured with the same ϵ. As an example, for the wc98 experiment with a moderate value

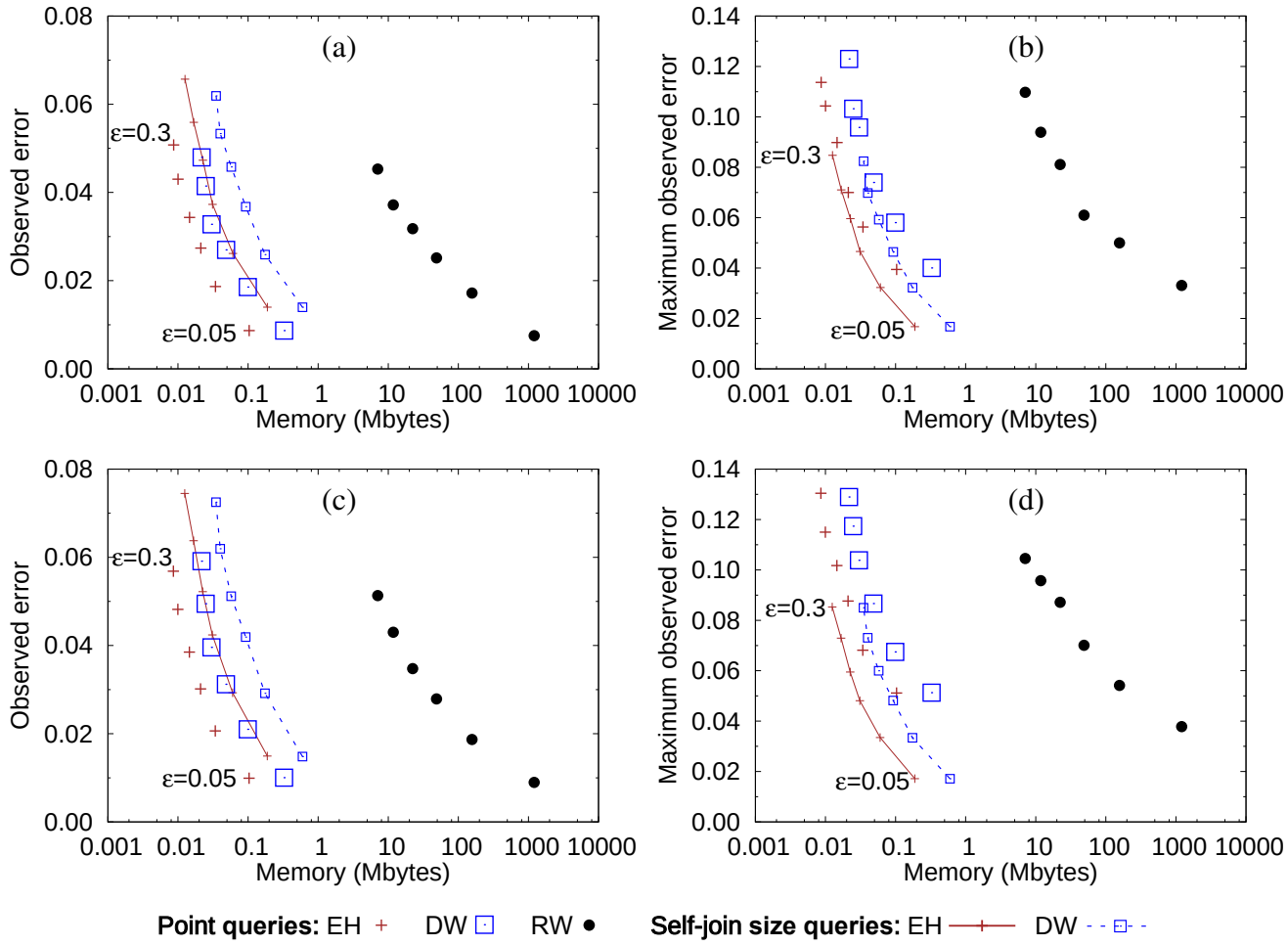

Fig. 5 Average and maximum observed error in correlation to memory requirements for a centralized setup: (a)-(b) wc98 data set, (c)-(d) caida data set. The points correspond to $\epsilon \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$.

of $\epsilon = 0.15$, the cost of maintaining the ECM-RW sketch is 48.8 Mbytes, whereas ECM-sketches based on exponential histograms and deterministic waves require 22 Kbytes and 50 Kbytes respectively. This happens because the memory requirements of randomized waves grow quadratically with $1/\epsilon$, whereas the two deterministic sliding window algorithms scale linearly. Note that this negative result applies to all known randomized sliding window algorithms, e.g., [35, 14], since they all scale quadratically with $1/\epsilon$. As such, ECM-sketches based on deterministic structures are more applicable for scenarios with hardware with less memory, like sensor networks and network devices. Also note that ECM-RW are substantially slower than ECM-EH and ECM-DW, supporting two orders of magnitude lower update and query rates (cf. Table 3).

Focusing on the two structures with deterministic sliding windows, we see that ECM-EH sketches are substantially more compact, requiring around one third of the space of ECM-DW for the same ϵ value. Concerning computational performance, both structures can support comparable update and query execution rates (ECM-DW is slightly faster than ECM-EH, mainly due to its $O(d)$ worst-case complexity per update, compared to $O(d \log N)$ for ECM-EH). The results are consistent for both data sets.

Summarizing, these first results demonstrate the superiority of ECM-EH and ECM-DW compared to ECM-RW, both in terms of compactness and computational performance. ECM-EH and ECM-DW have comparable computational performance, whereas in terms of compactness ECM-EH substantially outperforms ECM-DW.

7.3 Distributed Setup

The second series of experiments was designed to evaluate the suitability of ECM-sketches for distributed setups, and precisely: (a) for setups requiring one-time merging of ECM-sketches, possibly even in a hierarchical fashion, and

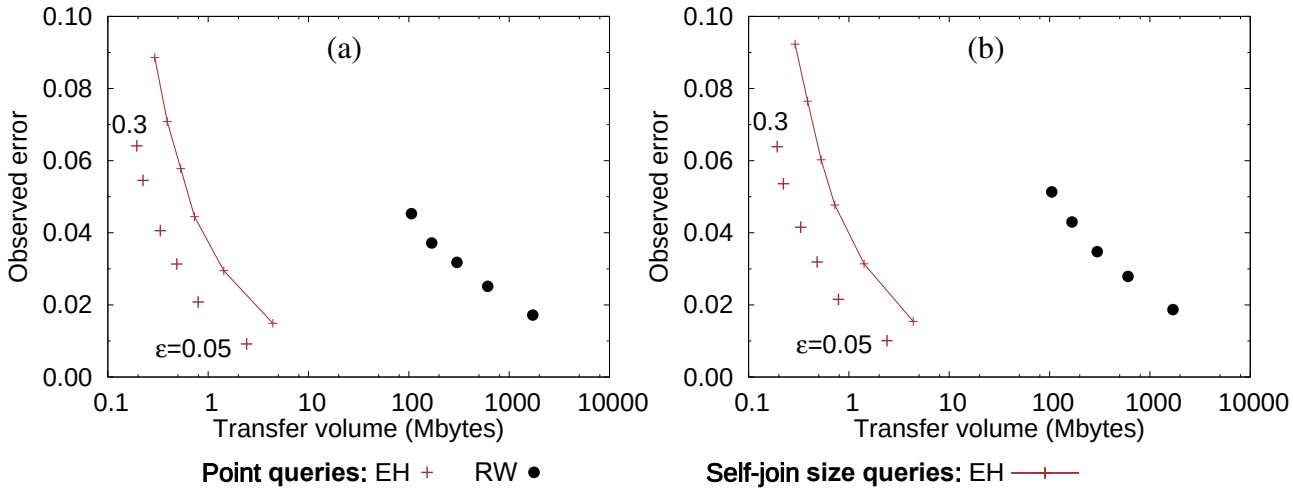

Fig. 6 Observed error in correlation to the network cost, for varying ϵ: (a) wc98 data set, (b) caida data set.

(b) for setups requiring continuous monitoring of functions through distributed ECM-sketches.

7.3.1 One-time Merging

These experiments focused on studying the influence of the network size and ϵ on the network cost. We have simulated a fixed network of $n \in [2, 256]$ sites, organized in an architecture resembling a balanced binary tree of height $\lceil \log_2(n) \rceil$. All sites resided at the leaf nodes of the tree, and were assigned the task of summarizing disjoint streams with ECM-sketches. Some of the sites were also randomly placed at the internal tree nodes, and were responsible for merging the sketches coming from their children. After completion of the streams, the sites pushed the resulting ECM-sketches to the root through the hierarchy, with merging at each intermediary node. At the end of this process, the root node of the hierarchy was holding a single ECM-sketch that represented the order-preserving merging of the n streams. ECM-DW sketches are not considered in this set of experiments, since they do not offer advantages compared to ECM-EH sketches with respect to compactness or accuracy.

Figures 6(a)-(b) plot the average observed error for point and self-join size queries in correlation to the network requirements for the whole merging process to be completed. The results correspond to a fixed network of 16 sites, with $\epsilon \in [0.05, 0.3]$ and $\delta = 0.15$. (Note that the simulation with ECM-RW sketches did not complete for $\epsilon = 0.05$ values, due to insufficient memory resources at the machine simulating the sites.) To illustrate the accuracy loss due to this merging, Table 4 presents a comparison between the observed error of the centralized and the distributed ECM-sketches.

As expected, the process of iterative mergings causes an inflation of the observed error for ECM-EH sketches. This inflation, however, is very small, and substantially lower than the theoretical worst-case bound derived by the analysis. For example, for the wc98 dataset with $\epsilon = 0.1$, the average observed error after all mergings is 0.020, whereas the centralized ECM-EH has an observed error of 0.018, i.e., the error inflation caused by the iterative ECM-EH mergings is less than 1/8 of the experimentally derived error of the centralized sketch. Concerning ECM-RW sketches, there is no systemic variation of the error, since randomized waves enable lossless merging at the expense of a larger memory footprint. However, the network required for performing this merging using ECM-RW is at least three orders of magnitude higher. This requirement is prohibitive for a large set of application scenarios, like sensor and mobile networks, where high network usage causes severe battery drainage.

To explore the influence of the network size on the estimation accuracy and network cost, we have also simulated networks of n sites, with $n = \{2, 4, \ldots, 256\}$. (For the case of ECM-RW, the number of sites reached only up to 64 due to memory constraints at the machine executing the simulation.) The sites were again placed as leaf nodes on a balanced binary tree, and updates were assigned to the sites randomly, with equal probability. Figures 7(a) and (c) plot the average observed error in correlation to the network size, for $\epsilon = \delta = 0.15$. As expected, for ECM-EH sketches, increasing the number of sites leads to a small increase on the observed estimation error, whereas the accuracy of ECM-RW sketches remains unaffected. However, similar to the previous experiment, the network cost for merging the sketches based on randomized waves (Figures 7(b) and (d)) is three orders of magnitude higher compared to ECM-EH. This limits the applicability of ECM-RW to cases where fast, fixed network is available, and makes the ability to merge deterministic sliding windows, e.g., based on exponential histograms, a very important contribution of this work.

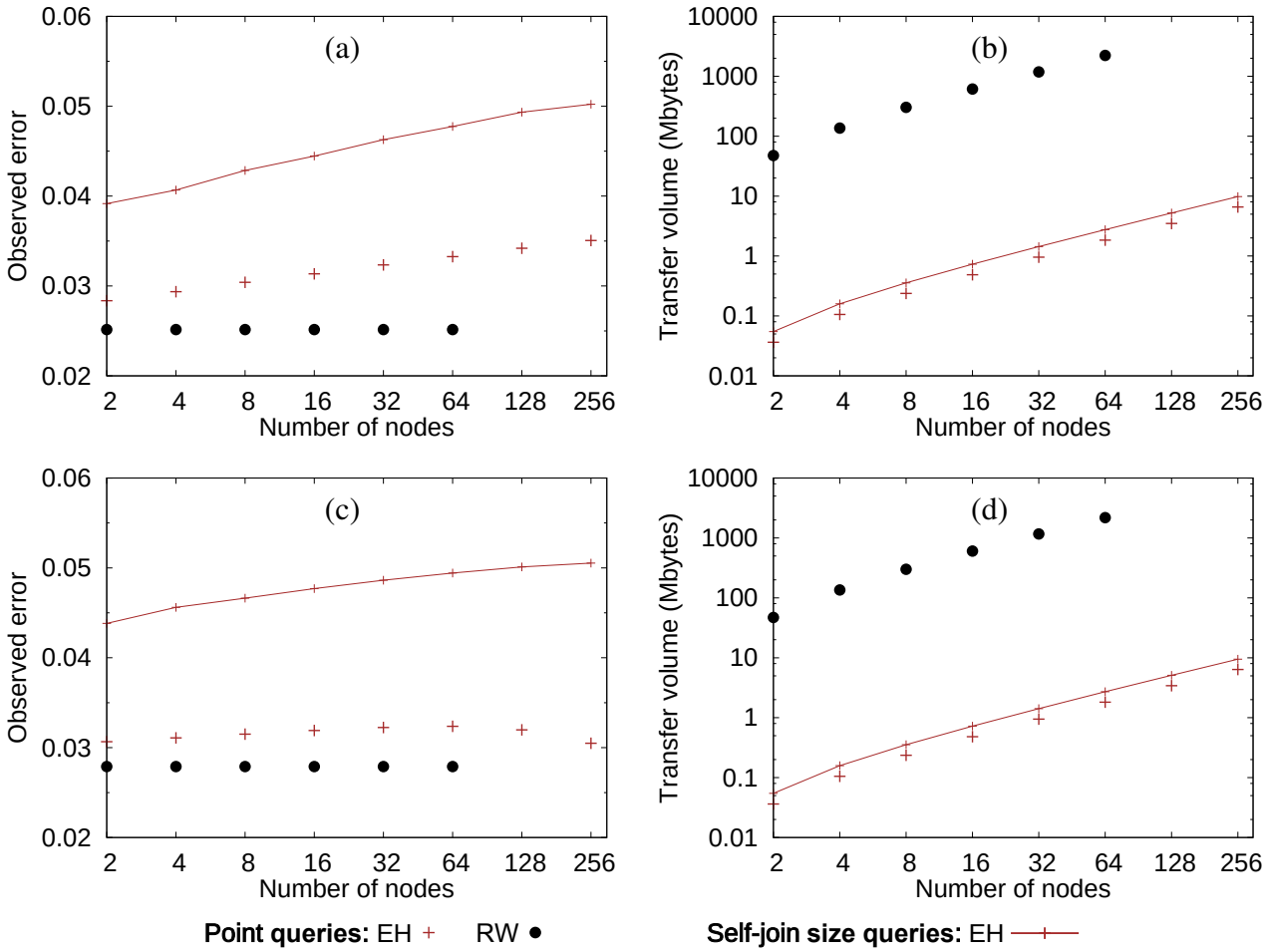

Fig. 7 Observed error and network cost for different network sizes: (a)-(b) wc98, (c)-(d) caida.

Summarizing, this set of experiments showed that ECM-sketches based on exponential histograms can be merged with very small information loss. Compared to the lossless merging of ECM-sketches based on randomized waves, ECM-EH are substantially more compact, and are therefore applicable for a wider range of application scenarios, where network cost and/or memory is of the essence, such as P2P networks, sensor networks, and network routers.

7.3.2 Continuous Monitoring

The final set of experiments investigates the suitability of ECM-sketches in combination with the geometric method for distributed continuous monitoring, as discussed in Section 6 (denoted as $\mathcal{A}_{ecm}$ hereafter). Particularly, we consider monitoring of the self-join size of a high-dimensional vector $\mathbf{v}$ that corresponds to sliding window statistics (i.e., item frequencies) aggregated over n data streams $S_1, S_2, \ldots, S_n$. Each stream S_i is monitored by site p_i, and all sites are enabled direct communication with a coordinator. Estimating the self-join size of such high-dimensional vectors is frequently useful, e.g., for query optimization in distributed databases, data partitioning, and computing a variety of useful indexes for streams (see [1] for a discussion). We only consider ECM-sketches constructed with exponential histograms, since these offer the best trade-off between memory and accuracy. As a baseline, we use the centralized algorithm (denoted as $\mathcal{A}_{cen}$), which relies on a central coordinator for collecting all updates from the remote sites and maintaining the accurate self-join size. Notice that $\mathcal{A}_{cen}$ has several practical considerations besides the high network cost, i.e., the coordinator still needs to efficiently maintain the high-dimensional statistics over a sliding window, which is challenging to achieve without ECM-sketches. Yet, we ignore this issue for our experiments. Both algorithms were allowed a warm-up phase (until the sliding window filled up for first time) before starting to measure cost and quality.

Figure 8(a) presents the transfer volume required by $\mathcal{A}_{ecm}$ for different network sizes as a ratio of the corresponding transfer volume of $\mathcal{A}_{cen}$. The results correspond to a configuration of $\mathcal{A}_{ecm}$ with $\delta = \epsilon = \theta = 0.15$. Clearly, $\mathcal{A}_{ecm}$ is substantially more efficient than the baseline, enabling net-

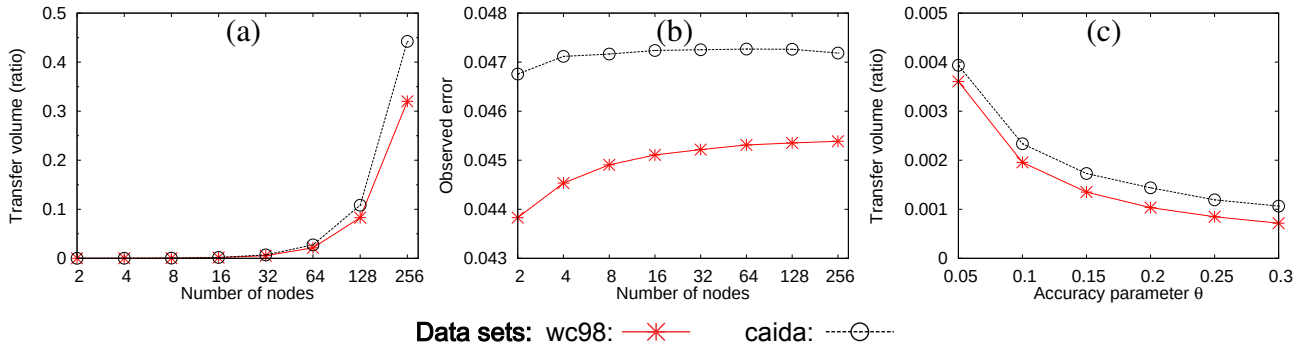

Fig. 8 (a)-(b) Network cost and observed error for different network sizes, (c) Effect of the value of θ.

work savings of up to two orders of magnitude for networks up to 32 sites. As expected, increasing the network size leads to an increase of the communication cost of $\mathcal{A}_{ecm}$ (the cost of $\mathcal{A}_{cen}$ does not change). This is a known characteristic of the geometric method. Nevertheless, even for the network of 256 sites, $\mathcal{A}_{ecm}$ still requires less than half the cost of $\mathcal{A}_{cen}$. We also see that caida is slightly more difficult to monitor compared to wc98. This is because wc98 is more stable than caida, i.e., as soon as the sliding window is filled, self-join size changes very slowly. Caida data set, on the other hand, is by nature more dynamic, causing more frequent threshold violations, and a higher network cost.

The average observed error for the same runs is shown in Figure 8(b). Even though error slightly increases with network size, the increase is negligible, and the error always remains smaller than the value of parameter θ, i.e., the error tolerance of the geometric method. All results are consistent for both data sets.

We have also tested the sensitivity of $\mathcal{A}_{ecm}$ on parameter θ. The results in Figure 8(c) correspond to a fixed network of 16 sites, with δ and ϵ set to 0.15. As expected, increasing θ drastically reduces the network cost of the algorithm: for a higher θ, $\mathcal{A}_{ecm}$ causes less threshold crossings, requiring less synchronizations of both phases. As an indication, for the caida dataset and for $\theta = 0.05$, $\mathcal{A}_{ecm}$ required 7133 first-phase synchronizations (i.e., synchronizations on $d_u|d_l$ only) and 2694 second-phase synchronizations (on the full sketches). For $\theta = 0.3$, these synchronizations were reduced to 5637 for the first phase, and only 457 for the second phase.

Summarizing, the experiments have shown that the combination of ECM-sketches with the geometric method can be used for efficiently monitoring of non-linear functions, such as the self-join size, in distributed settings. The network savings are substantial compared to the baseline algorithm that forwards all updates to a central coordinator, and typically exceed two orders of magnitude for small networks, whereas the observed error is negligible.

8 Conclusions

In this work we considered the problem of answering complex queries over distributed and high dimensional data streams, in the sliding window model. Our proposal, ECM-sketches, is a compact structure combining the state-of-the-art sketching technique for data stream summarization with deterministic sliding window synopses. The structure provides probabilistic accuracy guarantees for the quality of the estimation, for point queries and self-join size queries, and can enable a broad range of problems, such as finding heavy hitters, computing quantiles, and answering range queries over sliding windows.

Focusing on distributed applications, we also showed how a set of ECM-sketches, each one representing an individual stream, can be merged to generate a single ECM-sketch that summarizes the stream produced by the order-sensitive merging of all individual streams. Interestingly, this is the first result in the literature enabling such merging for deterministic sliding window synopses (or sketches based on these), and it is of high importance since deterministic synopses are generally a factor of $O(\log(1/\delta)/\epsilon)$ more compact than the best-known randomized synopsis for delivering an ϵ-accurate approximation. In the same context, we demonstrated how ECM-sketches can be exploited within the geometric method for answering continuous queries defined over sliding windows.

ECM-sketches were thoroughly evaluated with a set of extensive experiments, using two massive real-world datasets, and considering both centralized and distributed setups. The results verified the high performance of the structure. Compared to structures based on randomized sliding window synopses, ECM-sketches improve the memory and computational complexity by at least one order of magnitude. The same magnitude of improvement is observed with respect to the network requirements.

Our future work will focus on further optimizations for continuous distributed queries. Two interesting open problems include considering other query types, and concurrently

executing multiple continuous queries. In Section 6 we have already discussed initial optimizations for concurrent execution of many queries. We expect that both computation and network complexity can be reduced further by coordinating the synchronization process between the queries, and taking the accuracy requirements of each query into account during the synchronization process.

Acknowledgments. This work was supported by the European Commission under ICT-FP7-LEADS-318809 (Large-Scale Elastic Architecture for Data-as-a-Service) and ICT-FP7-FERARI-619491 (Flexible Event pRocessing for big dAta aRchItectures).

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. J. Comput. Syst. Sci. **58**(1), 137–147 (1999)
2. Arlitt, M., Jin, T.: A workload characterization study of the 1998 world cup web site. Network **14**(3), 30 –37 (2000)
3. Busch, C., Tirthapura, S.: A deterministic algorithm for summarizing asynchronous streams over a sliding window. In: STACS, pp. 465–476 (2007)
4. Chakrabarti, A., Cormode, G., Mcgregor, A.: A near-optimal algorithm for estimating the entropy of a stream. ACM Trans. Algorithms **6**(3), 51:1–51:21 (2010)
5. Chan, H.L., Lam, T.W., Lee, L.K., Ting, H.F.: Continuous monitoring of distributed data streams over a time-based sliding window. Algorithmica **62**(3-4), 1088–1111 (2012)
6. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: ICALP, pp. 693–703 (2002)
7. Cohen, E., Strauss, M.J.: Maintaining time-decaying stream aggregates. J. Algorithms **59**(1), 19–36 (2006)
8. Cormode, G., Garofalakis, M.: Approximate continuous querying over distributed streams. ACM Trans. Database Syst. **33**(2) (2008)
9. Cormode, G., Garofalakis, M., Muthukrishnan, S., Rastogi, R.: Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In: SIGMOD, pp. 25–36 (2005)
10. Cormode, G., Muthukrishnan, S.: What's hot and what's not: Tracking most frequent items dynamically. In: PODS, pp. 296–306 (2003)
11. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
12. Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Optimal sampling from distributed streams. In: PODS, pp. 77–86 (2010)
13. Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Continuous sampling from distributed streams. J. ACM **59**(2), 10:1–10:25 (2012)
14. Cormode, G., Tirthapura, S., Xu, B.: Time-decaying sketches for robust aggregation of sensor data. SIAM J. Comput. **39**(4), 1309–1339 (2009)
15. Cormode, G., Yi, K.: Tracking distributed aggregates over time-based sliding windows. In: SSDBM, pp. 416–430 (2012)
16. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM J. Comput. **31**(6), 1794–1813 (2002)
17. Dimitropoulos, X.A., Stoecklin, M.P., Hurley, P., Kind, A.: The eternal sunshine of the sketch data structure. Computer Networks **52**(17), 3248–3257 (2008)
18. Garofalakis, M.N., Keren, D., Samoladas, V.: Sketch-based geometric monitoring of distributed stream queries. PVLDB **6**(10), 937–948 (2013)
19. Gibbons, P.B.: Distinct sampling for highly-accurate answers to distinct values queries and event reports. In: VLDB, pp. 541–550 (2001)
20. Gibbons, P.B.: Distinct-values estimation over data streams. In: Data Stream Management: Processing High-Speed Data Streams. Springer (2007)
21. Gibbons, P.B., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: SPAA, pp. 63–72 (2002)
22. Greenwald, M.B., Khanna, S.: Space-efficient online computation of quantile summaries. In: SIGMOD, pp. 58–66 (2001)
23. Huang, L., Garofalakis, M., Joseph, A., Taft, N.: Communication efficient tracking of distributed cumulative triggers. In: ICDCS (2007)
24. Huang, L., Nguyen, X., Garofalakis, M., Hellerstein, J., Jordan, M., Joseph, A., Taft, N.: Communication-efficient online detection of network-wide anomalies. In: INFOCOM, pp. 134–142 (2007)
25. Hung, R.Y.S., Ting, H.F.: Finding heavy hitters over the sliding window of a weighted data stream. In: LATIN, pp. 699–710 (2008)
26. Jain, A., Hellerstein, J.M., Ratnasamy, S., Wetherall, D.: A wakeup call for internet monitoring systems: The case for distributed triggers. In: SIGCOMM Workshop on Hot Topics in Networks (HotNets) (2004)
27. Keren, D., Sharfman, I., Schuster, A., Livne, A.: Shape sensitive geometric monitoring. TKDE **24**(8), 1520–1535 (2012)
28. Mirkovic, J., Prier, G., Reiher, P.L.: Attacking DDoS at the source. In: ICNP, pp. 312–321 (2002)
29. Muthukrishnan, S.: "Data Streams: Algorithms and Applications". Foundations and Trends in Theoretical Computer Science **1**(2) (2005)
30. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: SIGMOD, pp. 563–574 (2003)
31. Papapetrou, O., Garofalakis, M.N., Deligiannakis, A.: Sketch-based querying of distributed sliding-window data streams. PVLDB **5**(10), 992–1003 (2012)
32. Qiao, L., Agrawal, D., El Abbadi, A.: Supporting sliding window queries for continuous data streams. In: SSDBM, pp. 85–96 (2003)
33. Sharfman, I., Schuster, A., Keren, D.: A geometric approach to monitoring threshold functions over distributed data streams. In: SIGMOD, pp. 301–312 (2006)
34. Tirthapura, S., Xu, B., Busch, C.: Sketching asynchronous streams over a sliding window. In: PODC, pp. 82–91 (2006)
35. Xu, B., Tirthapura, S., Busch, C.: Sketching asynchronous data streams over sliding windows. Distributed Computing **20**(5), 359–374 (2008)

Appendix

A Proofs for centralized queries

Lemmas 3 and 4 provide error guarantees for point and inner product queries on ECM-sketches, for any set of ϵ_{cm}, ϵ_{sw}, δ_{cm} and δ_{sw}. With Theorems 2 and 3 we derive the optimal values of these parameters (the ones that minimize the total cost), given only the acceptable total ϵ and δ.

Lemma 3 *With probability at least* $1 - \delta_{cm} - \delta_{sw}$,

$$|\hat{f}(x,r) - f(x,r)| \leq \begin{cases} (1+\epsilon_{sw})\epsilon_{cm}||a_r||_1 & \text{if } \epsilon_{sw} \leq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}, \\ \epsilon_{sw}||a_r||_1 & \text{if } \epsilon_{sw} \geq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}. \end{cases}$$

Proof We start with an overview of the proof. The ECM-sketch estimation is susceptible to two types of errors. The first is due to the hash collisions, i.e., two different items may hash to the same ECM-sketch cell. This error is relative to the L1 norm. The second is due to the sliding window counters, and is relative to the counter value, i.e., the number of items hashed to the particular counter. For the lemma, we derive a single error relative to the L1 norm by considering worst-case bounds (i.e., maximum possible values) for the combination of the two errors. We first bound the number of hash collisions that occur at counter $(j, h_j(x))$ for any row $j \leq d$ within the query range r, assuming that the sliding window algorithm offers perfect accuracy. The proof stems from the accuracy proof for Count-min sketches [11], differentiating on the estimation step of the number of hash collisions (by offering an error relative to $||a_r||_1 - f(x,r)$, instead of $||a_r||_1$). Then, we consider the error caused by the sliding window estimation.

Error due to hash collisions. Temporarily assume that the sliding window structure enables perfect accuracy (the assumption will be lifted later). With $I_{x,j,y}$ we denote the indicator variables which are 1 if $x \neq y \wedge h_j(x) = h_j(y)$, and 0 otherwise. We further define the variables $X_{x,j,r}$ to be $X_{x,j,r} = \sum_{y \in \mathcal{D}} I_{x,j,y} f(y,r)$. By our assumption that the sliding window estimation is accurate, $E(j, h_j(x), r) = f(x,r) + X_{x,j,r}$. Since the ECM-sketch will return $\hat{f}(x,r) = \min_j E(j, h_j(x), r) = f(x,r) + \min_j X_{x,j,r}$ as a frequency estimate, the estimation error will be $\hat{f}(x,r) - f(x,r) = \min_j X_{x,j,r}$, which can be bounded as follows.

By pairwise independence of the hash functions: $E(I_{x,j,y}) = Pr[h_j(x) = h_j(y)] \leq 1/w = \frac{\epsilon}{e}$. Therefore, the expected value of $X_{x,j,r}$ is $E(X_{x,j,r}) = E\left(\sum_{k=1}^{n} I_{x,j,y} f(k,r)\right) = \sum_{\forall k \in \mathcal{D} \setminus \{x\}} f(k,r) E(I_{x,j,y}) \leq (||a_r||_1 - f(x,r))\frac{\epsilon}{e}$. Furthermore, by Markov inequality:

$$
\begin{aligned}
&Pr[\min_j X_{x,j,r} > \epsilon(||a_r||_1 - f(x,r))] = \\
&Pr[\forall j : X_{x,j,r} > \epsilon(||a_r||_1 - f(x,r))] \leq \\
&Pr[\forall j : X_{x,j,r} > eE(X_{x,j,r})] < e^{-d} \leq \delta_{cm} \qquad (1)
\end{aligned}
$$

Error due to the sliding window estimation. In practice, the sliding window algorithm may introduce errors to the computation of $E(j, h_j(x), r)$. Let $R(j, h_j(x), r)$ denote the accurate number of bits contained within the query range at counter $(j, h_j(x))$. Then, an (ϵ, δ)-approximate sliding window algorithm guarantees that $Pr[E(j, h_j(x), r) - R(j, h_j(x), r) \leq \epsilon_{sw} R(j, h_j(x), r)] \geq 1 - \delta_{sw}$.

Consider row $j = \min\arg_j E(j, h_j(x), r)$, i.e., the row with $E(j, h_j(x), r) = \hat{f}(x,r)$. For the case that $\hat{f}(x,r) > R(j, h_j(x), r)$, we have:

$$
\begin{aligned}
&Pr[\hat{f}(x,r) \leq (1+\epsilon_{sw}) R(j, h_j(x), r)] \geq 1 - \delta_{sw} \Rightarrow \\
&Pr[\hat{f}(x,r) \leq (1+\epsilon_{sw})(f(x,r) + X_{x,j,r})] \\
&\qquad \geq 1 - \delta_{sw} \Rightarrow \\
&Pr[\hat{f}(x,r) - f(x,r) \leq X_{x,j,r} + \epsilon_{sw}(f(x,r) + X_{x,j,r}) \\
&\qquad \geq 1 - \delta_{sw}
\end{aligned}
$$

Furthermore, $X_{x,j,r}$ can be bounded by Inequality 1, giving: $Pr[\hat{f}(x,r) - f(x,r) \leq \epsilon_{cm}(||a_r||_1 - f(x,r)) + \epsilon_{sw}(f(x,r) + \epsilon_{cm}(||a_r||_1 - f(x,r)))] \geq 1 - \delta_{sw} - \delta_{cm}$. For convenience we define $c = f(x,r)/||a_r||_1$. Then, $Pr[\hat{f}(x,r) - f(x,r) \leq ||a_r||_1(\epsilon_{cm}(1-c) + \epsilon_{sw}(c + \epsilon_{cm}(1-c)))] = Pr[\hat{f}(x,r) - f(x,r) \leq ||a_r||_1(c(\epsilon_{sw} - \epsilon_{cm} - \epsilon_{sw}\epsilon_{cm}) + \epsilon_{cm} + \epsilon_{sw}\epsilon_{cm})] \geq 1 - \delta_{sw} - \delta_{cm}$.

Variable c takes values between 0 and 1 (inclusive). When $\epsilon_{sw} \leq \epsilon_{cm}/(1-\epsilon_{cm})$, the RHS of the inequality (the error) is maximized for $c = 0$. Otherwise, the RHS is maximized for $c = 1$. Therefore, with a probability of at least $1 - \delta_{cm} - \delta_{sw}$:

$$
\hat{f}(x,r) - f(x,r) \leq \begin{cases} ||a_r||_1 \epsilon_{cm}(1+\epsilon_{sw}) & \text{if } \epsilon_{sw} \leq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}, \\ ||a_r||_1 \epsilon_{sw} & \text{if } \epsilon_{sw} \geq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}. \end{cases} \qquad (2)
$$

With a similar analysis, the case of $\hat{f}(x,r) < R(j, h_j(x), r)$ gives a much tighter constraint:

$$
Pr[f(x,r) - \hat{f}(x,r) \leq \epsilon_{sw} f(x,r)] \geq 1 - \delta_{sw} \qquad (3)
$$

The lemma follows directly by inequalities 2 and 3. □

Theorem 3

Proof We first consider an ECM-sketch with a deterministic sliding window structure, e.g., an exponential histogram. We want to derive the combination of ϵ_{cm} and ϵ_{sw} that minimizes the space complexity of the sketch for a given ϵ, i.e., minimizes $C(\epsilon) = O(\frac{\ln(1/\delta_{cm})}{\epsilon_{cm}\epsilon_{sw}})$. We study the two cases of Lemma 3 separately:

Case 1 ($\epsilon_{sw} \leq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}$): We first exploit the fact that

$$
\epsilon = (1+\epsilon_{sw})\epsilon_{cm} \qquad (4)
$$

to eliminate ϵ_{sw} from the space complexity of the sketch:

$$
C(\epsilon) = O\left(\frac{\ln(1/\delta_{cm})}{\epsilon_{sw}\epsilon_{cm}}\right) = O\left(\frac{\ln(1/\delta_{cm})}{\frac{\epsilon - \epsilon_{cm}}{\epsilon_{cm}}\epsilon_{cm}}\right) = O\left(\frac{\ln(1/\delta_{cm})}{\epsilon - \epsilon_{cm}}\right)
$$

The cost is minimized when the denominator is maximized. For a fixed ϵ chosen by the user, this happens when ϵ_{cm} is minimized. The smallest ϵ_{cm} satisfying Eqn. 4 and the precondition of Case 1 is $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$, resulting to $\epsilon_{sw} = \epsilon$.

Case 2 ($\epsilon_{sw} \geq \frac{\epsilon_{cm}}{1-\epsilon_{cm}}$): By Lemma 3, we see that setting $\epsilon_{sw} = \epsilon$ we achieve the required accuracy. In order to derive ϵ_{cm}, notice that we want to minimize the complexity $C(\epsilon) = O(\frac{\ln(1/\delta_{cm})}{\epsilon_{cm}\epsilon_{sw}})$. This is achieved by maximizing ϵ_{cm}. The maximum value of ϵ_{cm} satisfying the precondition of Case 2 is $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$.

Notice that both cases lead to the same combination for minimizing the cost, i.e., $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$ and $\epsilon_{sw} = \epsilon$.

The case of randomized waves is similar. The cost function becomes $O(\ln(1/\delta_{cm})\ln(1/\delta_{sw})/(\epsilon_{cm}\epsilon_{sw}^2))$, with the constraint that $\delta = \delta_{cm} + \delta_{sw}$. The cost is minimized for $\delta_{cm} = \delta_{sw} = \delta/2$, $\epsilon_{cm} = \frac{\epsilon}{1+\epsilon}$ and $\epsilon_{sw} = \epsilon$. □

Lemma 4 *With probability at least* $1 - \delta_{cm}$,

$$
|\widehat{a_r \odot b_r} - a_r \odot b_r| \leq \begin{cases} ||a_r||_1||b_r||_1 \epsilon_{cm}(1+\epsilon_{sw})^2 \\ \qquad \text{if } \epsilon_{cm} \geq \frac{\epsilon_{sw}^2 + 2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}, \\ ||a_r||_1||b_r||_1(\epsilon_{sw}^2 + 2\epsilon_{sw}) \\ \qquad \text{if } \epsilon_{cm} \leq \frac{\epsilon_{sw}^2 + 2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}. \end{cases}
$$

Proof We first examine the case that $\widehat{a_r \odot b_r} > a_r \odot b_r$. Consider the estimation derived by any single row j of the ECM-sketch. With $E_a(i,j,r)$ we denote the frequency estimation of the sliding window counter at position (i,j) for stream a and for query range r.

$$E(\widehat{(a_r \odot b_r)}_j - a_r \odot b_r) = \sum_{i=1}^{w} E_a(i,j,r)E_b(i,j,r) - a_r \odot b_r$$

$$\leq \sum_{i=1}^{w} \sum_{\substack{p\in\mathcal{D},\\ h_j(p)=i}} f_a(p,r) \sum_{\substack{q\in\mathcal{D},\\ h_j(q)=i}} f_b(q,r) * (1+\epsilon_{sw})^2 - a_r \odot b_r$$

$$= \sum_{i=1}^{w} \sum_{\substack{x\in\mathcal{D},\\ h_j(x)=i}} f_a(x,r)f_b(x,r) * (1+\epsilon_{sw})^2 +$$

$$\sum_{i=1}^{w} \sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)=i}} f_a(p,r)f_b(q,r) * (1+\epsilon_{sw})^2 - a_r \odot b_r$$

$$= (1+\epsilon_{sw})^2 \Big(\sum_{x\in\mathcal{D}} f_a(x,r)f_b(x,r) +$$

$$\sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)}} f_a(p,r)f_b(q,r)\Big) - a_r \odot b_r$$

$$= a_r \odot b_r(\epsilon_{sw}^2 + 2\epsilon_{sw}) +$$

$$\sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)}} f_a(p,r)f_b(q,r)(1+\epsilon_{sw})^2 \quad (5)$$

Our next step is to bound $\sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)}} f_a(p,r)f_b(q,r)$. For convenience we use $X_{i,j,r}$ as a shortcut for $\sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)}} f_a(p,r)f_b(q,r)$. Then,

$$E(X_{i,j,r}) = \sum_{p,q\in\mathcal{D},p\neq q} Pr[h_j(p)=h_j(q)] f_a(p,r)f_b(q,r)$$

$$= \frac{1}{w} \sum_{p,q\in\mathcal{D},p\neq q} f_a(p,r)f_b(q,r)$$

$$\leq \frac{\epsilon_{cm}}{e}\Big(\sum_{p,q\in\mathcal{D}} f_a(p,r)f_b(q,r) - a_r \odot b_r\Big)$$

$X_{i,j,r}$ can be bounded by Markov inequality:

$$Pr[\min_j X_{i,j,r} > \epsilon_{cm}(\sum_{p,q\in\mathcal{D}} f_a(p,r)f_b(q,r) - a_r \odot b_r)] =$$

$$Pr[\forall j : X_{i,j,r} > eE(X_{i,j,r})] < e^{-d} \leq \delta_{cm} \quad (6)$$

Let $c = \frac{a_r \odot b_r}{||a_r||_1||b_r||_1}$. Combining Equation 5 and Inequality 6:

$$\widehat{a_r \odot b_r} - a_r \odot b_r$$

$$\leq a_r \odot b_r(\epsilon_{sw}^2 + 2\epsilon_{sw}) + \sum_{\substack{p,q\in\mathcal{D},p\neq q,\\ h_j(p)=h_j(q)}} f_a(p,r)f_b(q,r)(1+\epsilon_{sw})^2$$

$$< a_r \odot b_r(\epsilon_{sw}^2 + 2\epsilon_{sw}) + (1+\epsilon_{sw})^2 \min_j X_{i,j,r}$$

$$\leq a_r \odot b_r(\epsilon_{sw}^2 + 2\epsilon_{sw}) +$$

$$(1+\epsilon_{sw})^2\epsilon_{cm}\left(\sum_{p,q\in\mathcal{D}} f_a(p,r)f_b(q,r) - a_r \odot b_r\right)$$

$$= a_r \odot b_r(\epsilon_{sw}^2 + 2\epsilon_{sw}) + (1+\epsilon_{sw})^2\epsilon_{cm}(||a_r||_1||b_r||_1 - a_r \odot b_r)$$

$$= c||a_r||_1||b_r||_1(\epsilon_{sw}^2 + 2\epsilon_{sw}) + \epsilon_{cm}(1+\epsilon_{sw})^2||a_r||_1||b_r||_1(1-c)$$

$$= ||a_r||_1||b_r||_1\left(c(\epsilon_{sw}^2 + 2\epsilon_{sw}) + \epsilon_{cm}(1+\epsilon_{sw})^2(1-c)\right)$$

with probability at least $1-\delta_{cm}$.

The values of c that maximize the error (the RHS) are $c=1$ when $\epsilon_{cm} < \frac{\epsilon_{sw}^2+2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}$, and $c=0$ when $\epsilon_{cm} \geq \frac{\epsilon_{sw}^2+2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}$. The corresponding maximum errors are $||a_r||_1||b_r||_1(\epsilon_{sw}^2+2\epsilon_{sw})$ (for $c=1$), and $||a_r||_1||b_r||_1\epsilon_{cm}(1+\epsilon_{sw})^2$ (for $c=0$).

With a similar analysis, the case of $\widehat{a_r \odot b_r} < a_r \odot b_r$ gives a tighter constraint: $Pr[a_r \odot b_r - \widehat{a_r \odot b_r} > (\epsilon_{sw}^2+2\epsilon_{sw})a_r \odot b_r] < \delta_{sw}$. The lemma follows directly. □

Theorem 2

Proof Similar to the analysis for point queries, we need to consider the two cases of Lemma 4 separately.

Case 1 ($\epsilon_{cm} \geq \frac{\epsilon_{sw}^2+2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}$): By Lemma 4, we set $\epsilon_{cm}(1+\epsilon_{sw})^2 = \epsilon$ in order to achieve the required accuracy. The space complexity then becomes $C(\epsilon) = \frac{1}{\epsilon_{sw}\epsilon_{cm}} = \frac{(1+\epsilon_{sw})^2}{\epsilon\epsilon_{sw}}$. Since $\frac{(1+\epsilon_{sw})^2}{\epsilon\epsilon_{sw}}$ is strictly decreasing for ϵ_{sw} in the interval $[0,1]$, we can minimize the space complexity by setting the maximum value for ϵ_{sw} satisfying the case's precondition $\epsilon_{cm} \geq \frac{\epsilon_{sw}^2+2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}$, i.e., $\epsilon_{sw} = \sqrt{\epsilon+1}-1$. Then, ϵ_{cm} becomes equal to $\frac{\epsilon}{\epsilon+1}$.

Case 2 ($\epsilon_{cm} \leq \frac{\epsilon_{sw}^2+2\epsilon_{sw}}{(\epsilon_{sw}+1)^2}$): By Lemma 4, in order to achieve the required accuracy we need to set $\epsilon_{sw}^2 + 2\epsilon_{sw}^2 = \epsilon \Rightarrow \epsilon_{sw} = \sqrt{\epsilon+1}-1$. Accordingly, $\epsilon_{cm} = \frac{\epsilon}{\epsilon+1}$.

Notice that, similar to the point queries analysis, the two cases lead to the same configuration for minimizing the cost, i.e., $\epsilon_{cm} = \frac{\epsilon}{\epsilon+1}$ and $\epsilon_{sw} = \sqrt{\epsilon+1}-1$. □

B Proofs for distributed setups

Theorem 4 derives worst-case error bounds for the merging of exponential histograms. Lemma 2 and Theorem 5 prove the correctness of the algorithm for continuous self-join size queries.

Theorem 4

Proof We argue that $EH_\oplus$ approximates the exponential histogram of the logical stream, with a maximum relative error of $\epsilon + \epsilon' + \epsilon\epsilon'$, where ϵ is the error parameter of the initial exponential histograms. Consider a query for the last q time units. With $s_q = t - q$ we denote the query starting time. Let Q denote the index of the bucket of $EH_\oplus$ which contains s_q in its range, i.e., $s(EH_\oplus^Q) \leq s_q \leq e(EH_\oplus^Q)$. With i and $\hat{i}$ we denote the accurate and estimated number of true bits in the query range. According to the estimation algorithm, the estimation for the number of true bits in the stream will be $\hat{i} = 1/2|EH_\oplus^Q| + \sum_{1\leq Y<Q} |EH_\oplus^Y|$. This estimation may be influenced by two types of approximation errors: (a) a possible approximation error of the overlap of bucket $EH_\oplus^Q$ with the query range, denoted as err_1, and, (b) a possible approximation error of i, denoted as err_2, because of the inclusion of data that arrived before s_q in buckets $Y \leq Q$, or data that arrived after s_q in buckets $Y > Q$. Let us now look into these two errors in more details.

With respect to err_2, recall that the contents of individual buckets are inserted to $EH_\oplus$ using the starting time and the ending time of the buckets. Therefore, it may happen that some bits arrive before s_q but are inserted to $EH_\oplus$ with a timestamp after s_q, creating 'false positives'. The opposite is also possible. These bits are called out-of-order bits with respect to s_q. Clearly, out-of-order bits may lead to underestimation or overestimation of the query answer. According to Lemma 5, the number of out-of-order bits originating from each exponential histogram EH_x is at most ϵi_x, with i_x denoting the accurate number of true bits that were inserted in EH_x at or after s_q. The number of out-of-order bits from all streams is then bounded as follows: $\text{err}_2 \leq \sum_{x=1}^{n} \epsilon i_x = \epsilon \sum_{x=1}^{n} i_x = \epsilon i$.

Underestimation or overestimation of the overlap may also happen because of the halving of the size of bucket $EH_\oplus^Q$ during query time

(err_1). As shown in [16], this process may introduce a maximum relative error of ϵr, where r is the sum of the sizes of all buckets in $EH_{\oplus}$ with an index lower than Q (i.e., with a starting time at least equal to s_q). Recall that r may also include bits that have arrived before s_q (the out-of-order bits), which is however upper-bounded by ϵi, as discussed before. Therefore, the maximum underestimation or overestimation error is $\text{err}_1 = \epsilon' r \leq \epsilon'(i + \epsilon i) = \epsilon' i + \epsilon\epsilon' i$, with $i = \sum_{x=1}^{n} i_x$.

Summing err_1 and err_2, we get a maximum relative error of $(\epsilon + \epsilon' + \epsilon\epsilon')$, which completes the proof. □

Lemma 5 *Consider an individual exponential histogram EH_x of stream X, configured with error parameter ϵ. The out-of-order bits with respect to the query starting time s_q that EH_x can generate are at most ϵi_x, with i_x denoting the number of true bits arriving at or after s_q in X.*

Proof Due to the non-decreasing nature of bucket timestamps, there can be only one bucket with a start time less than s_q and end time greater than or equal to s_q. Let this bucket be EH_x^j. All other buckets have both starting and ending time at the same side of s_q, and therefore their contents are always inserted with a timestamp at the correct side of s_q and do not create out-of-order bits.

Since the ending time of EH_x^j is at or after s_q, its most recent true bit has arrived at or after s_q, and should be included in the query range. Therefore, the number of true bits arriving at or after s_q in stream X is $i_x \geq 1 + \sum_{b=1}^{j-1} |EH_x^b|$. Furthermore, since half of the bits of EH_x^j are inserted using the ending time and half using the starting time of the bucket, the maximum number of out-of-order bits is $|EH_x^j|/2$. By construction (invariant 1):

$$\frac{|EH_x^j|}{2(1 + \sum_{b=1}^{j-1} |EH_x^b|)} \leq \epsilon \Rightarrow \frac{|EH_x^j|}{2} \leq \epsilon(1 + \sum_{b=1}^{j-1} |EH_x^b|) \leq \epsilon i_x$$

□

Lemma 2

Proof The proof relies on the following properties of the min:

Monotonicity: If $\mathbf{x}[i] \leq \mathbf{y}[i]$ for all i, then $\min_i\{x[i]\} \leq \min_i\{y[i]\}$.

Distributivity: For any monotonically increasing function f, $\min_i\{f(\mathbf{x}[i])\} = f(\min_i\{\mathbf{x}[i]\})$.

We want to derive sufficient conditions such that $(1-\theta)\boldsymbol{f}(\mathbf{v}(t)) \leq \boldsymbol{f}(\mathbf{v}(t_0)) \leq (1+\theta)\boldsymbol{f}(\mathbf{v}(t))$, with $\boldsymbol{f}(\mathbf{v}(t)) = \min_{row=1}^{d}\{||\mathbf{v}[row]||^2\}$. By the distributivity property of the min for monotonically increasing functions (i.e., the square root), it is sufficient to verify:

$$\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1+\theta}} \leq \min_{row=1}^{d}\{||\mathbf{v}(t)[row]||\} \leq \sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}.$$

By the triangle inequality:

$$||\mathbf{v}(t)[row] - \mathbf{v}(t_0)[row]|| \leq \sum_{j=1}^{n} ||\mathbf{v}_j(t)[row] - \mathbf{v}_j(t_0)[row]||$$

$$= \sum_{j=1}^{n} \boldsymbol{d}_j[row] = n\boldsymbol{d}[row] \Rightarrow$$

$$||\mathbf{v}(t_0)[row]|| - n\boldsymbol{d}[row] \leq ||\mathbf{v}(t)[row]|| \leq ||\mathbf{v}(t_0)[row]|| + n\boldsymbol{d}[row] \quad (7)$$

Notice that $||\mathbf{v}(t_0)[row]||$ is constant per synchronization. Therefore, Inequality 7 bounds $||\mathbf{v}(t)[row]||$ by a linear relation of $\boldsymbol{d}$, i.e., it allows us to form threshold-crossing queries in the R^d space. By monotonicity of the min, it suffices to monitor the following conditions:

$$\min_{i=1}^{d}\{||\mathbf{v}(t_0)[i]|| + n\boldsymbol{d}[i]\} \leq \sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}$$

and

$$\min_{i=1}^{d}\{||\mathbf{v}(t_0)[i]|| - n\boldsymbol{d}[i]\} \geq \sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1+\theta}}.$$

The lemma follows directly, by dividing both sides of the conditions by n. □

Theorem 5

Proof Sketch: By construction, all counters of $\mathbf{v}_i^u(t)$ are at least equal to the corresponding counters of $\mathbf{v}_i(t)$. Therefore, the self-join size estimate for $\mathbf{v}_i^u(t)$ will be at least equal to the self-join size estimate for $\mathbf{v}_i(t)$ at all times. Using Lemma 2 to monitor $\mathbf{v}$ but only considering the shifts which increase the counters, we get that if $\min_{row=1}^{d}\{\frac{||\mathbf{v}(t_0)[row]||}{n} + \boldsymbol{d}^u[row]\} \leq \frac{1}{n}\sqrt{\frac{\boldsymbol{f}(\mathbf{v}(t_0))}{1-\theta}}$, then $\boldsymbol{f}(\mathbf{v}(t_0)) \leq (1+\theta)\boldsymbol{f}(\mathbf{v}(t))$. The lower bound is shown analogously. □