Sketching Streams Through the Net: Distributed Approximate Query Tracking

Graham Cormode
Bell Labs, Lucent Technologies
cormode@bell-labs.com

Minos Garofalakis[*]
Intel Research, Berkeley
minos@acm.org

Abstract

Emerging large-scale monitoring applications require continuous tracking of complex data-analysis queries over collections of physically-distributed streams. Effective solutions have to be simultaneously space/time efficient (at each remote monitor site), communication efficient (across the underlying communication network), and provide continuous, guaranteed-quality approximate query answers. In this paper, we propose novel algorithmic solutions for the problem of continuously tracking a broad class of complex aggregate queries in such a distributed-streams setting. Our tracking schemes maintain approximate query answers with provable error guarantees, while simultaneously optimizing the storage space and processing time at each remote site, and the communication cost across the network. They rely on tracking general-purpose randomized sketch summaries of local streams at remote sites along with concise prediction models of local site behavior in order to produce highly communication- and space/time-efficient solutions. The result is a powerful approximate query tracking framework that readily incorporates several complex analysis queries (including distributed join and multi-join aggregates, and approximate wavelet representations), thus giving the first known low-overhead tracking solution for such queries in the distributed-streams model.

1 Introduction

Traditional data-management applications typically require database support for a variety of *one-shot queries*, including lookups, sophisticated slice-and-dice operations, data mining tasks, and so on. One-shot means the data processing is essentially done once, in response to the posed query. This has led to a very successful industry of database engines optimized for supporting complex, one-shot SQL queries over large amounts of data. Recent years, however, have witnessed the emergence of a new class of *large-scale event monitoring* applications that pose novel data-management challenges. In one class of applications, monitoring a large-scale system is a crucial aspect of system operation and maintenance. As an example, consider the Network Operations Center (NOC) for the IP-backbone network of a large ISP (such as Sprint or AT&T). Such NOCs typically need to monitor hundreds or thousands of network elements (e.g., routers, links) and events at blistering speeds, continuously tracking and correlating data from a multitude of points in the network in order to quickly detect and react to hot spots, floods, element failures, and attacks. A different class of applications is one in which monitoring is the goal in itself. For instance, consider a wireless network of sensors deployed for habitat and environmental monitoring or inventory tracking. The key objective for such systems is to continuously monitor and correlate sensor measurements for trend analysis, detecting moving objects, intrusions, or other adverse events.

A closer examination of such monitoring applications allows us to abstract a number of common characteristics. First, monitoring is *continuous*, that is, we need real-time tracking of measurements or events, not merely one-shot responses to sporadic queries. Second, monitoring is inherently *distributed*, that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange information through a communication network. This also means that there typically are important *communication constraints* owing to either network-capacity restrictions (e.g., in IP-network monitoring, where the volumes of collected utilization and traffic data can be huge [7]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [18]). Furthermore, each remote site may see a *high-speed stream* of data and has its own local resource limitations, such as *storage-space* or *processing-time* constraints.

[*] Work done while at Bell Labs, Lucent Technologies.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

This is certainly true for IP routers (that cannot possibly store the log of all observed packet traffic at high network speeds), as well as wireless sensor nodes (that, even though they may not observe large data volumes, typically have very little memory onboard).

Another key aspect of large-scale event monitoring is the need for effectively tracking queries that *combine and/or correlate information* (e.g., IP traffic or sensor measurements) observed across the collection of remote sites. For instance, tracking the result size of a *join* (the "workhorse" correlation operator in the relational world) over the streams of fault/alarm data from two or more IP routers (e.g., with a join condition based on their observed timestamp values) can allow network administrators to effectively detect correlated fault events at the routers, and, perhaps, also pinpoint the *root-causes* of specific faults in real time. As another example, consider the tracking of a two- or three-dimensional histogram summary of the traffic-volume distribution observed across the edge routers of a large ISP network (along axes such as time, source/destination IP address, etc.); clearly, such a histogram could provide a valuable visualization tool for effective circuit provisioning, detection of anomalies and DoS attacks, and so on. Interestingly, when tracking statistical properties of large-scale systems, answers that are precise to the last decimal are typically not needed; instead, *approximate query answers* (with reasonable guarantees on the approximation error) are often sufficient, since we are typically looking for indicators or patterns, rather than precisely-defined events. This works in our favor, allowing us to effectively tradeoff efficiency with approximation quality.

Prior Work. Given the nature of large-scale monitoring applications, their importance for security as well as daily operations, and their general applicability, surprisingly little is known about solutions for many basic distributed-monitoring problems. The bulk of recent 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 [15], estimating distinct values [13], counting frequent elements (i.e., "heavy hitters") [4, 20], approximating large Haar-wavelet coefficients [14], and estimating join sizes and stream norms [1, 2, 11]. All the above methods work in a centralized, one-shot setting and, therefore, do not consider communication-efficiency issues. More recent work has proposed methods that carefully optimize site communication costs for approximating different queries in a distributed setting, including quantiles [16] and heavy hitters [19]; however, the underlying assumption is that the computation is triggered either periodically or in response to a one-shot request. Such techniques are not immediately applicable for *continuous-monitoring*, where the goal is to continuously provide real-time, guaranteed-quality estimates over a distributed collection of streams.

Closest in spirit to our work are the recent results of Olston et al. [3, 21], Das et al. [8], and our recent work on distributed quantile tracking [6]. All these efforts consider the tradeoff between accuracy and communication for monitoring a limited class of continuous queries (at a coordinator site) over distributed streams (at remote sites). More specifically, Olston et al. [3, 21] consider tracking approximate top-k values and simple aggregates (e.g., AVERAGE or MAX) over dynamically-changing numeric values spread over multiple sources, whereas Das et al. [8] discuss monitoring of approximate set-expression cardinalities over physically-distributed element streams. Similarly, our recent work [6] attacks the problem of approximately tracking *one-dimensional* quantile summaries of a global data distribution spread over the remote sites. All these earlier papers focus solely on a narrow class of distributed-monitoring queries (e.g., one-dimensional quantiles), resulting in special-purpose solutions applicable only to the specific form of queries at hand. It is not at all clear if/how they can be extended to more general settings (such as, tracking distributed *joins* or *multi-dimensional* data summaries).

Our Contributions. In this paper, we tackle the problem of continuously tracking approximate, guaranteed-quality answers to a *broad, general class of complex aggregate queries* over a collection of distributed data streams. Our contributions are as follows.

- **Communication- and Space-Efficient Approximate Query Tracking.** We present the first known algorithms for tracking a broad class of complex data-analysis queries over a distributed collection of streams to specified accuracy, provably, at all times. In a nutshell, our tracking algorithms achieve communication and space efficiency through a combination of general-purpose *randomized sketches* for summarizing local streams, and concise *sketch-prediction models* for capturing the update-stream behavior at local sites. The use of prediction models, in particular, allows our schemes to achieve a natural notion of *stability*, rendering communication unnecessary as long as local data distributions remain stable (or, *predictable*). The end result is a powerful, general-purpose approximate query tracking framework that readily incorporates several complex data-analysis queries (including join and multi-join aggregates, and approximate wavelet/histogram representations in one or more dimensions), thus giving the first principled, low-overhead tracking solution for such queries in the distributed-streams model. In fact, as our analysis demonstrates, the worst-case communication cost for simple cases of our protocols is comparable to that of a one-shot computation, while their space requirement is not much higher than that of centralized, one-shot estimation methods for data streams.

- **Time-Efficient Sketch-Tracking Algorithms, and Extensions to Other Streaming Models.** When dealing with massive, rapid-rate data streams (e.g., monitoring high capacity network links), the *time* needed to process each update (e.g., to maintain a sketch summary of the stream) becomes a critical concern. Traditional approaches that need to "touch" every part of the sketch summary can quickly become infeasible. The problem is further compounded in our tracking schemes that need to continuously track the divergence of the sketch from an evolving sketch prediction. We address this problem by proposing a novel structure for randomized sketches that allows us to *guarantee small (i.e., logarithmic) update and tracking times* (regardless of the size of the sketch), while offering the same (in fact, slightly improved) space/accuracy tradeoffs. Furthermore, we discuss the extension of our distributed-tracking schemes and results to different data-streaming models that place more emphasis on recent updates to the stream (using either *sliding-window* or *exponential-decay* mechanisms).

• **Experimental Results Validating our Approach.** We perform a thorough set of experiments with our schemes over real-life data to verify their benefits in practical scenarios. The results clearly demonstrate that our algorithms can result in dramatic savings in communication — reducing overall communication costs by a factor of more than 20 for an approximation error of only 10%. The use of sophisticated, yet concise, sketch-prediction models is key to obtaining the best results. Furthermore, our numbers show that our novel schemes for fast local sketch updates and tracking can allow each remote site to process many hundreds of thousands of updates per second, matching even the highest-speed data streams.

Throughout, we have chosen to omit all proof arguments due to space constraints.

2 Preliminaries

System Architecture. We consider a distributed-computing environment, comprising a collection of k *remote sites* and a designated *coordinator site*. Streams of data updates arrive continuously at remote sites, while the coordinator site is responsible for generating approximate answers to (possibly, continuous) user queries posed over the *unions* of remotely-observed streams (across all sites). Following earlier work in the area [3, 6, 8, 21], our distributed stream-processing model does not allow direct communication between remote sites; instead, as illustrated in Figure 1, a remote site exchanges messages only with the coordinator, providing it with state information on its (locally-observed) streams. Note that such a hierarchical processing model is, in fact, representative of a large class of applications, including network monitoring where a central Network Operations Center (NOC) is responsible for processing network traffic statistics (e.g., link bandwidth utilization, IP source-destination byte counts) collected at switches, routers, and/or Element Management Systems (EMSs) distributed across the network.

Each remote site $j \in \{1, \ldots, k\}$ observes local update streams that incrementally render a collection of (up to) s distinct *frequency distribution vectors* (equivalently, multisets) $\boldsymbol{f}_{1,j}, \ldots, \boldsymbol{f}_{s,j}$ over data elements from corresponding integer domains $[U_i] = \{0, \ldots, U_i - 1\}$, for $i = 1, \ldots, s$; that is, $\boldsymbol{f}_{i,j}[v]$ denotes the frequency of element $v \in [U_i]$ observed locally at remote site j. As an example, in the case of IP routers monitoring the number of TCP connections and UDP packets exchanged between source and destination IP addresses, $[U_1] = [U_2]$ denote the domain of 64-bit (source, destination) IP-address pairs, and $\boldsymbol{f}_{1,j}$, $\boldsymbol{f}_{2,j}$ capture the frequency of specific (source, destination) pairs observed in TCP connections and UDP packets routed through router j. (We use $\boldsymbol{f}_{i,j}$ to denote both the i^{th} update stream at site j as well as the underlying element multiset/frequency distribution in what follows.) Each stream update at remote site j is a triple of the form $< i, v, \pm 1 >$, denoting an insertion ("$+1$") or deletion ("-1") of element $v \in [U_i]$ in the $\boldsymbol{f}_{i,j}$ frequency distribution (i.e., a change of ± 1 in v's net frequency in $\boldsymbol{f}_{i,j}$). All frequency distribution vectors $\boldsymbol{f}_{i,j}$ in our distributed streaming architecture change dynamically over time — when necessary, we make this dependence explicit, using $\boldsymbol{f}_{i,j}(t)$ to denote the state of the vector at time t (assuming a consistent notion of "global time" in our distributed system). (The unqualified notation $\boldsymbol{f}_{i,j}$ typically refers to the *current* state of the frequency vector.)

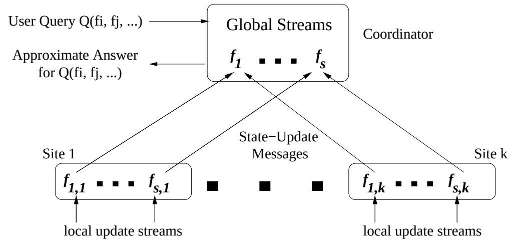


Figure 1: Distributed Stream Processing Architecture.

Problem Formulation. For each $i \in \{1, \ldots, s\}$, we define the *global* frequency distribution vector $\boldsymbol{f}_i$ for the i^{th} update stream as the summation of the corresponding local, per-site vectors; that is, $\boldsymbol{f}_i = \sum_{j=1}^{k} \boldsymbol{f}_{i,j}$. Note that, in general, the local sub-streams for a stream $\boldsymbol{f}_i$ may only be observed at a *subset* of the k remote sites – we use $\texttt{sites}(\boldsymbol{f}_i)$ to denote that subset, and write $k_i = |\texttt{sites}(\boldsymbol{f}_i)|$ (hence $k_i \le k$). Our focus is on the problem of effectively answering user queries over this collection of global frequency distributions $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_s$ at the coordinator site. Rather than one-time query evaluation, we assume a continuous-querying environment which implies that the coordinator needs to *continuously maintain* (or, *track*) the approximate answers to user queries as the local update streams $\boldsymbol{f}_{i,j}$ evolve at individual remote sites. More specifically, we focus on a broad class of user queries $Q = Q(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_s)$ over the global frequency vectors, including:

• *Inner- and Tensor-Product Queries (i.e., Join and Multi-Join Aggregates).* Given a pair of global frequency vectors $\boldsymbol{f}_1$, $\boldsymbol{f}_2$ over the same data domain $[U]$, the *inner-product query* $Q(\boldsymbol{f}_1, \boldsymbol{f}_2) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2 = \sum_{v=0}^{U-1} \boldsymbol{f}_1[v] \cdot \boldsymbol{f}_2[v]$ is the result size of an (equi)join query over the corresponding streams (i.e., $|\boldsymbol{f}_1 \bowtie \boldsymbol{f}_2|$). More general, *tensor product queries* $Q(\boldsymbol{f}_i, \boldsymbol{f}_l, \boldsymbol{f}_m, \ldots) = \boldsymbol{f}_i \cdot \boldsymbol{f}_l \cdot \boldsymbol{f}_m \cdots$ over multiple (domain-compatible) frequency vectors $\boldsymbol{f}_i$, $\boldsymbol{f}_l$, $\boldsymbol{f}_m, \ldots$ capture the result size of the corresponding multi-join query $\boldsymbol{f}_i \bowtie \boldsymbol{f}_l \bowtie \boldsymbol{f}_m \cdots$ (see, e.g., [11]); here the notion of a "frequency vector" is generalized to capture a (possibly) *multi-dimensional* frequency distribution (i.e., a *tensor*). For instance, in the three-way join query $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \cdot \boldsymbol{f}_3 = \sum_u \sum_v \boldsymbol{f}_1[u] \cdot \boldsymbol{f}_2[u,v] \cdot \boldsymbol{f}_3[v]$, the $\boldsymbol{f}_2$ vector captures the joint distribution of the two attributes of stream $\boldsymbol{f}_2$ participating in the join. Without loss of generality, we continue to view such multi-dimensional frequency tensors as vectors (e.g., assuming some standard linearization of the tensor entries, such as row-major). In the relational world, join and multi-join queries are basically the "workhorse" operations for correlating two or more data sets. Thus, they play a crucial role in any kind of data analysis over multiple data collections. Our discussion here focuses primarily on join and multi-join result sizes (i.e., COUNT aggregates), since our approach and results extend to other aggregate functions in a relatively straightforward manner (as discussed in [11]).

• *L_2-Norm Queries (i.e., Self-Join Sizes).* The *self-join size* query for a (global) stream $\boldsymbol{f}_i$ is defined as the square of the L_2 norm ($||\cdot||$) of the corresponding frequency vector; that is, $Q(\boldsymbol{f}_i) = ||\boldsymbol{f}_i||^2 = \boldsymbol{f}_i \cdot \boldsymbol{f}_i = \sum_v (\boldsymbol{f}_i[v])^2$. The self-join size represents important demographic information about a data collection; for instance, its value is an indication of the degree of skew in the data [2].

• *Range Queries, Point Queries, and Heavy Hitters.* A *range query* with parameters $[a, b]$ over a frequency distribution $\boldsymbol{f}_i$ is the sum of the values of the distribution in the given range; that is, $R(\boldsymbol{f}_i, a, b) = \sum_{v=a}^{b} \boldsymbol{f}_i[v]$. A *point query* is the special case of a range query when $a = b$. The *heavy hitters* are those points $v \in U_i$ satisfying $R(\boldsymbol{f}_i, v, v) \geq \phi \cdot R(\boldsymbol{f}_i, 0, U_i - 1)$ (i.e., their frequency exceeds a ϕ-fraction of the overall number of stream elements) for a given $\phi < 1$ [4, 5].

• *Histogram and Wavelet Representations.* A histogram query $H(\boldsymbol{f}_i, B)$ or wavelet query $W(\boldsymbol{f}_i, B)$ over a frequency distribution $\boldsymbol{f}_i$ asks for a B-bucket histogram representation, or a B-term (Haar) wavelet representation of the $\boldsymbol{f}_i$ vector, respectively. The goal is to minimize the error of the resulting approximate representation, typically defined as the L_2 norm of the difference between the $H(\boldsymbol{f}_i, B)$ or $W(\boldsymbol{f}_i, B)$ approximation and either the true distribution $\boldsymbol{f}_i$, or the *best-possible* B-term representation of $\boldsymbol{f}_i$ [14, 22].

The distributed nature of the local streams comprising the global frequency distributions $\{\boldsymbol{f}_i\}$ raises difficult algorithmic challenges for our approximate query tracking problems. Naïve schemes that accurately track query answers by forcing remote sites to ship every remote stream update to the coordinator are clearly impractical, since they not only impose an inordinate burden on the underlying communication infrastructure (especially, for high-rate data streams and large numbers of remote sites), but also drastically limit the battery life of power-constrained remote devices (such as wireless sensor nodes) [10, 18]. A main part of our approach is to adopt the paradigm of continuous tracking of *approximate* query answers at the coordinator site with strong guarantees on the quality of the approximation. This allows our schemes to effectively trade-off communication efficiency and query-approximation accuracy in a precise, quantitative manner; in other words, larger error tolerances for the approximate answers at the coordinator imply smaller communication overheads to ensure continuous approximate tracking.

Randomized Stream Sketching. Techniques based on small-space pseudo-random *sketch* summaries of the data have proved to be very effective tools for dealing with massive, rapid-rate data streams in a centralized setting [2, 1, 5, 14, 11]. The key idea in such sketching techniques is to represent a streaming frequency vector $\boldsymbol{f}$ using a much smaller *sketch* vector (denoted by $\mathtt{sk}(\boldsymbol{f})$) that can be easily maintained as the updates incrementally rendering $\boldsymbol{f}$ are streaming by. Typically, the entries of the sketch vector $\mathtt{sk}(\boldsymbol{f}))$ are appropriately-defined *random variables* with some desirable properties that can provide probabilistic guarantees for the quality of the data approximation.

More specifically, consider the AGMS (or, "tug-of-war") sketches proposed by Alon, Gibbons, Matias, and Szegedy in their seminal papers [2, 1]:[1] The i^{th} entry in an AGMS sketch $\mathtt{sk}(\boldsymbol{f})$ is defined as the random variable $\sum_{v=0}^{U-1} \boldsymbol{f}[v] \cdot \xi_i[v]$, where $\{\xi_i[v] : v \in [U]\}$ is a family of four-wise independent binary random variables uniformly distributed in $\{-1, +1\}$ (with mutually-independent families used across different entries of the sketch). The key here is that, using appropriate pseudo-random hash functions, each such family can be efficiently constructed online in small (i.e., $O(\log U)$) space [2]. Note that, by construction, each entry of $\mathtt{sk}(\boldsymbol{f})$ is essentially a *randomized linear projection* (i.e., an inner product) of the $\boldsymbol{f}$ vector (using the corresponding ξ family), that can be easily maintained over the input update stream: Start with each counter $\mathtt{sk}(\boldsymbol{f})[i] = 0$ and, for each i, simply set $\mathtt{sk}(\boldsymbol{f})[i] = \mathtt{sk}(\boldsymbol{f})[i] + \xi_i[v]$ ($\mathtt{sk}(\boldsymbol{f})[i] = \mathtt{sk}(\boldsymbol{f})[i] - \xi_i[v]$) whenever an insertion (resp., deletion) of v is observed in the stream. Another critical property is the *linearity* of such sketch structures: Given two "parallel" sketches (built using the same ξ families) $\mathtt{sk}(\boldsymbol{f}_1)$ and $\mathtt{sk}(\boldsymbol{f}_2)$ and scalars α, β, then $\mathtt{sk}(\alpha\boldsymbol{f}_1 + \beta\boldsymbol{f}_2) = \alpha\mathtt{sk}(\boldsymbol{f}_1) + \beta\mathtt{sk}(\boldsymbol{f}_2)$ (i.e., the sketch of a linear combination of streams is simply the linear combination of their individual sketches). The following theorem summarizes some of the basic estimation properties of AGMS sketches (for centralized streams) that we employ in our study. (Throughout, the notation $x \in (y \pm z)$ is equivalent to $|x - y| \leq z$.)

Theorem 2.1 ([1, 2]). *Let $\mathtt{sk}(\boldsymbol{f}_1)$ and $\mathtt{sk}(\boldsymbol{f}_2)$ denote two parallel sketches comprising $O(\frac{1}{\epsilon^2}\log(1/\delta))$ counters, built over the streams $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$, where ϵ, $1 - \delta$ denote the desired bounds on error and probabilistic confi-*

[1] Our techniques and results can also be extended to other randomized stream sketching methods, such as the *Count-Min sketches* [5]; due to space constraints, details are omitted.

dence, respectively. Then, with probability at least $1-\delta$, $||\mathtt{sk}(\boldsymbol{f}_1) - \mathtt{sk}(\boldsymbol{f}_2)||^2 \in (1 \pm \epsilon)||\boldsymbol{f}_1 - \boldsymbol{f}_2||^2$ *and* $\mathtt{sk}(\boldsymbol{f}_1) \cdot \mathtt{sk}(\boldsymbol{f}_2) \in (\boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \pm \epsilon||\boldsymbol{f}_1||||\boldsymbol{f}_2||)$*. The processing time required to maintain each sketch is* $O(\frac{1}{\epsilon^2}\log(1/\delta))$ *per update.*

Thus, the self-join of the difference of the sketch vectors gives a high-probability, ϵ relative-error estimate of the self-join of the difference of the actual streams (so, naturally, $||\mathtt{sk}(\boldsymbol{f}_1)||^2 \in (1 \pm \epsilon)||\boldsymbol{f}_1||^2$); similarly, the inner product of the sketch vectors gives a high-probability estimate of the join of the two streams to within an additive error of $\epsilon||\boldsymbol{f}_1||||\boldsymbol{f}_2||$. [2] To provide ϵ relative-error guarantees for the binary join query $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$, Theorem 2.1 can be applied with error bound $\epsilon' = (\epsilon||\boldsymbol{f}_1||||\boldsymbol{f}_2||)/(\boldsymbol{f}_1 \cdot \boldsymbol{f}_2)$, giving a total sketching space requirement of $O(\frac{||\boldsymbol{f}_1||^2||\boldsymbol{f}_2||^2}{\epsilon^2(\boldsymbol{f}_1\cdot\boldsymbol{f}_2)^2}\log(1/\delta))$ counters [1].

The results in Theorem 2.1 can be extended in a natural manner to the case of multi-join aggregate queries [11]: Given an m-way join (i.e., tensor-product) query $Q(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \cdots \boldsymbol{f}_m$, and corresponding parallel AGMS sketch vectors $\mathtt{sk}(\boldsymbol{f}_1), \ldots, \mathtt{sk}(\boldsymbol{f}_m)$ of size $O(\frac{1}{\epsilon^2}\log(1/\delta))$ (built based on the specific join predicates in the query [11]), the inner product of the sketches $\Pi_{i=1}^m \mathtt{sk}(\boldsymbol{f}_i)$ can be shown to be within an additive error of $\epsilon(2^{m-1}-1)^2\Pi_{i=1}^m||\boldsymbol{f}_i||$ of the true multi-join result size. The full development can be found in [11].

3 Our Query-Tracking Solution

The goal of our tracking algorithms is to ensure strong error guarantees for approximate answers to queries over the collection of global streams $\{\boldsymbol{f}_i : i = 1, \ldots, s\}$ at the coordinator, while minimizing the amount of communication with the remote sites. We can also identify other important design desiderata that our solution should strive for: (1) *Minimal global information exchanges* — schemes in which the coordinator distributes information on the *global* streams to remote sites would typically need to rebroadcast up-to-date global information to sites (either periodically or during some "global resolution" stage [3, 8]) to ensure correctness; instead, our solutions are designed to explicitly avoid such expensive "global synchronization" steps; (2) *Summary-based information exchange* — rather than shipping complete update streams $\boldsymbol{f}_{i,j}$ to the coordinator, remotes sites only communicate concise summary information (e.g., sketches) on their locally-observed updates; and, (3) *Stability* — intuitively, the stability property means that, provided the behavior of the local streams at remote sites remains reasonably stable (or, *predictable*), there is no need for communication between the remote sites and the coordinator.

Our solution avoids global information exchange entirely by each individual remote site j continuously monitoring only the L_2 norms of its *local* update streams $\{\boldsymbol{f}_{i,j} : i = 1, \ldots, s\}$. When a certain amount of change is observed locally, then a site may send a concise *state-update* message in order to update the coordinator with more recent information about its local update stream, and then resumes monitoring its local updates (Figure 1). Such state-update messages typically comprise a small sketch summary of the offending local stream(s) (along with, possibly, additional summary information), to allow the coordinator to continuously maintain accurate approximate answers to user queries. Our tracking scheme depends on two parameters ϵ and θ, where: ϵ captures the error of the local sketch summaries communicated to the coordinator; and, θ captures (an upper bound on) the deviation of the local-stream L_2 norms at each remote site involved in the query since the last communication with the coordinator. The overall error guarantee provided at the coordinator is given by a function $g(\epsilon, \theta)$, depending on the specific form of the query being tracked. It is important to note, however, that the local constraints at each remote site are essentially identical (i.e., simply tracking L_2-norm deviations for individual streams), *regardless* of the specific (global) query being tracked; as our results demonstrate, the combination of small sketch summaries and local constraints on the stream L_2 norms at individual sites is sufficient to provide high-probability error guarantees for a *broad class of queries* over the global streams $\{\boldsymbol{f}_i : i = 1, \ldots, s\}$. To the best of our knowledge, this work is the first to provide such a *general-purpose* distributed-tracking mechanism for approximate query answers.

Intuitively, larger θ values allow for larger local deviations since the last communication and, so, imply fewer communications to the coordinator. But, for a given error tolerance, the size of the ϵ-approximate sketches sent during each communication is larger (since $g(\epsilon, \theta)$ is increasing in both parameters). We provide some analysis that allows us to optimally divide the allowed query-error tolerance in simple cases, and provide empirical guidelines for more complex scenarios based on our experimental observations.

A local sketch summary $\mathtt{sk}(\boldsymbol{f}_{i,j}(t))$ communicated to the coordinator gives an (ϵ-approximate) picture of the snapshot of the $\boldsymbol{f}_{i,j}$ stream at time t.[3] To achieve *stability*, a crucial component of our solutions are concise *sketch-prediction models* that may be communicated from remote sites to the coordinator (along with the local stream summaries) in an attempt to accurately capture the anticipated behavior of local streams. The key idea here is to enable each site j and the coordinator to share a prediction of how the stream $\boldsymbol{f}_{i,j}$ evolves over time. The coordinator employs

[2] We note that the above "inner product" operator over sketch vectors is slightly more complex, involving both averaging and median-selection operations over the sketch-vector components [1, 2]. — formally, each sketch vector can be viewed as a two-dimensional $n \times m$ array, where $n = O(\frac{1}{\epsilon^2})$, $m = O(\log(1/\delta))$, and the "inner product" in the sketch-vector space for both the join and self-join case is defined as

$$\mathtt{sk}(\boldsymbol{f}_1) \cdot \mathtt{sk}(\boldsymbol{f}_2) = \underset{j=1,\ldots,m}{\text{median}} \{\frac{1}{n}\sum_{i=1}^{n} \mathtt{sk}(\boldsymbol{f}_1)[i,j] \cdot \mathtt{sk}(\boldsymbol{f}_2)[i,j]\}.$$

[3] To simplify the exposition, we assume that communications with the coordinator are instantaneous. In the case of non-trivial delays in the underlying communication network, techniques based on time-stamping and message serialization can be employed to ensure correctness, as in [21].

this prediction to answer user queries, while the remote site checks that the prediction is close (within θ bounds) to the actual observed distribution $\boldsymbol{f}_{i,j}$. As long the prediction accurately captures the local update behavior at the remote site, no communication is needed. Taking advantage of the linearity properties of sketch summaries allows us to represent the predicted distribution using a concise *predicted sketch*; thus, our predictions are also based solely on concise summary information that can be efficiently exchanged between remote site and coordinator when the model is changed. A high-level schematic of our distributed tracking scheme is depicted in Figure 2. The key insight from our results is that, as long as local constraints are satisfied, the predicted sketches at the coordinator are basically equivalent to $g(\epsilon, \theta)$*-approximate sketch summaries* of the global data streams.

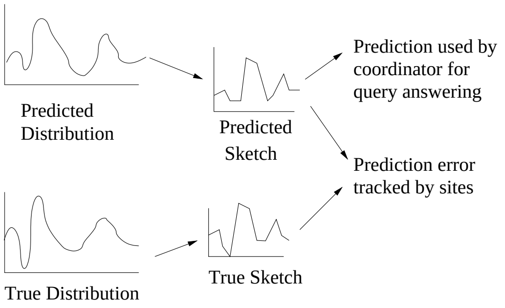


Figure 2: Schematic of Sketch-Prediction-Based Tracking.

In the remainder of this section, we discuss the details of our distributed query-tracking schemes, and our proposed sketch-prediction models for capturing remote-site behavior. In addition, we introduce a simple, yet very effective, improvement of the basic AGMS sketching technique that plays a crucial role in allowing remote sites to track their local constraints over massive, rapid-rate streams in *guaranteed small time* per update.

3.1 The Basic Tracking Scheme

We present our tracking scheme focusing primarily on inner-product and generalized, tensor-product (i.e., multi-join) queries, since our results for the other query classes discussed in Section 2 follow as corollaries of the inner-product case (Section 3.4). We focus on a single *inner-product (i.e., join) query* $Q(\boldsymbol{f}_1, \boldsymbol{f}_2) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2$ over our distributed-tracking architecture. Consider a remote site j participating in the distributed evaluation of $Q(\boldsymbol{f}_1, \boldsymbol{f}_2)$ (i.e., $j \in \texttt{sites}(\boldsymbol{f}_1) \cup \texttt{sites}(\boldsymbol{f}_2)$) — we assume that each such site maintains AGMS sketches on its locally observed sub-streams $\boldsymbol{f}_{1,j}$ and/or $\boldsymbol{f}_{2,j}$. (we often omit the “AGMS” qualification in what follows). If each participating site sends the coordinator its *up-to-date* local-stream sketches $\texttt{sk}(\boldsymbol{f}_{1,j}(t))$ and/or $\texttt{sk}(\boldsymbol{f}_{2,j}(t))$, then, by sketch linearity, the coordinator can simply compute the up-to-date sketches of the global streams $\texttt{sk}(\boldsymbol{f}_i(t)) = \sum_j \texttt{sk}(\boldsymbol{f}_{i,j}(t))$ $(i = 1, 2)$, and provide an approximate answer to the join query at time t with the error guarantees specified in Theorem 3.5.[4]

In our tracking scheme, to minimize the overall communication overhead, remote sites can also potentially ship a concise *sketch-prediction model* for their local updates to $\boldsymbol{f}_i$ (in addition to their local-stream sketches) to the coordinator. The key idea behind a sketch-prediction model is that, in conjunction with the communicated local-stream sketch, it allows the coordinator to construct a *predicted* $\texttt{sk}^p(\boldsymbol{f}_{i,j}(t))$ for the up-to-date state of the local-stream sketch $\texttt{sk}(\boldsymbol{f}_{i,j}(t))$ at any future time instant t, based on the locally-observed update behavior at the remote site. The coordinator then employs these collections of *predicted sketches* $\texttt{sk}^p(\boldsymbol{f}_{i,j})$ to continuously track an approximate answer to the distributed-join query. (We discuss different options for sketch-prediction models in Section 3.2). Fix a site $j \in \texttt{sites}(\boldsymbol{f}_i)$ (where $i \in \{1, 2\}$). After shipping its local sketch $\texttt{sk}(\boldsymbol{f}_{i,j})$ and (possibly) a corresponding sketch-prediction model to the coordinator, site j continuously monitors the L_2 norm of the deviation of its local, up-to-date sketch $\texttt{sk}(\boldsymbol{f}_{i,j}(t))$ from the corresponding predicted sketch $\texttt{sk}^p(\boldsymbol{f}_{i,j}(t))$ employed for estimation at the coordinator. The site checks the following condition at every time instant t:

$$||\texttt{sk}(\boldsymbol{f}_{i,j}(t)) - \texttt{sk}^p(\boldsymbol{f}_{i,j}(t))|| \le \frac{\theta}{\sqrt{k_i}} ||\texttt{sk}(\boldsymbol{f}_{i,j}(t))|| \quad (*)$$

that is, a communication to the coordinator is triggered only if the relative L_2-norm deviation of the local, up-to-date sketch $\texttt{sk}(\boldsymbol{f}_{i,j}(t))$ from the corresponding predicted sketch exceeds $\frac{\theta}{\sqrt{k_i}}$ (recall, $k_i = |\texttt{sites}(\boldsymbol{f}_i)|$). The pseudo-code for processing stream updates and tracking local constraints at remote sites, as well as providing approximate answers at the coordinator is depicted in Figure 3. The following theorem demonstrates that, as long as the local L_2-norm deviation constraints are met at all participating sites for the distributed $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$ join, then we can provide strong error guarantees for the approximate query answer (based on the predicted sketches) at the coordinator.

Theorem 3.1. *Assume local-stream sketches of size* $O(\frac{1}{\epsilon^2}\log(1/\delta))$*, and let* $\hat{s_i} = \sum_{j \in \texttt{sites}(\boldsymbol{f}_i)} \texttt{sk}^p(\boldsymbol{f}_{i,j})$ *(*$i \in \{1, 2\}$*). Also, assume that, for each remote site* $j \in \texttt{sites}(\boldsymbol{f}_i)$ *(*$i \in \{1, 2\}$*), the condition (*) is satisfied. Then, with probability at least* $1 - 2(k_1 + k_2)\delta$*,*

$$\hat{s_1} \cdot \hat{s_2} \in \boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \pm (\epsilon + (1+\epsilon)^2((1+\theta)^2 - 1))||\boldsymbol{f}_1||||\boldsymbol{f}_2||.$$

In other words, using local sketches of size $O(\frac{1}{\epsilon^2}\log(\frac{k_1+k_2}{\delta}))$, satisfying the local L_2-norm deviation constraints at each participating remote site ensures that the approximate answer for the join size $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$ computed using only the predicted sketches at the coordinator is within an absolute error of $\pm g_Q(\epsilon, \theta)||\boldsymbol{f}_1||||\boldsymbol{f}_2||$ of the

[4] This also assumes an initial “coordination” step where each remote site obtains the size parameters for its local sketches and the corresponding hash functions (same across all sites) from the coordinator.

```
Procedure SiteUpdate($j, i, v, \pm 1, \epsilon, \delta, \theta, k_i$)
Input: Site index $j$, stream index $i$, inserted/deleted value $v \in [U]$;
  sketch error, confidence, and local-deviation parameters $\epsilon, \delta, \theta$;
  "distribution factor" $k_i$ for stream $i$.
UpdateSketch($\mathtt{sk}(\boldsymbol{f}_{i,j}), < i, v, \pm 1 >$)    //update current and
UpdatePredictedSketch($\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$)    //predicted sketches
if $||\mathtt{sk}(\boldsymbol{f}_{i,j}) - \mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))|| > \frac{\theta}{\sqrt{k_i}} ||\mathtt{sk}(\boldsymbol{f}_{i,j})||$ then
    Compute sketch-prediction model $\mathtt{predModel}(\boldsymbol{f}_{i,j})$
    Send $\{i, j, \mathtt{sk}(\boldsymbol{f}_{i,j}), \mathtt{predModel}(\boldsymbol{f}_{i,j})\}$ to coordinator

Procedure EstimateJoin($\mathtt{id}(\boldsymbol{f}_1), \mathtt{id}(\boldsymbol{f}_2)$)
Input: Global-stream identifiers $\mathtt{id}(\boldsymbol{f}_1), \mathtt{id}(\boldsymbol{f}_2)$.
Output: Approximate answer to join-size query $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$.
for $i := 1$ to $2$ do
    Set $\mathtt{sk}^p(\boldsymbol{f}_i(t)) := 0$
    for each $j \in \mathtt{sites}(\boldsymbol{f}_i)$ do
        $\mathtt{sk}^p(\boldsymbol{f}_i(t)) := \mathtt{sk}^p(\boldsymbol{f}_i(t)) + \mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$
return $\mathtt{sk}^p(\boldsymbol{f}_1(t)) \cdot \mathtt{sk}^p(\boldsymbol{f}_2(t))$
```

Figure 3: Procedures for (a) Sketch Maintenance and Tracking at Remote Site $j \in \mathtt{sites}(\boldsymbol{f}_i)$ ($i \in \{1, 2\}$), and (b) Join-Size Estimation at the Coordinator. (t denotes current time)

exact answer. Note that these error guarantees are very similar to those obtained for the much simpler, centralized case (Theorem 2.1), with the only difference being the approximation-error bound of $g_Q(\epsilon, \theta) = \epsilon + (1+\epsilon)^2 ((1+\theta)^2 - 1) \approx \epsilon + 2\theta$ (ignoring quadratic terms in ϵ, θ which are typically very small since $\epsilon, \theta \ll 1$). The following corollary gives the adaptation of our tracking result for the special case of a *self-join* query $Q(\boldsymbol{f}_1) = ||\boldsymbol{f}_1||^2 = \sum_v (\boldsymbol{f}_1[v])^2$.

Corollary 3.2. *Assume local-stream sketches of size $O(\frac{1}{\epsilon^2}\log(1/\delta))$, and let $\hat{s_1} = \sum_{j \in \mathtt{sites}(\boldsymbol{f}_1)} \mathtt{sk}^p(\boldsymbol{f}_{1,j})$. If each remote site $j \in \mathtt{sites}(\boldsymbol{f}_1)$ satisfies the condition (*), then with probability at least $1 - 2k_i\delta$, $||\hat{s_1}||^2 \in [1 \pm (\epsilon + (1+\epsilon)^2 ((1+\theta)^2 - 1))] ||\boldsymbol{f}_1||^2 \approx (1 \pm (\epsilon + 2\theta)) ||\boldsymbol{f}_1||^2$.*

Extension to Multi-Joins. The analysis and results for our distributed-tracking scheme can also be extended to the case of distributed *multi-join* (i.e., tensor-product) queries. More formally, consider an m-way distributed join $Q(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \cdots \boldsymbol{f}_m$ and corresponding parallel sketches $\mathtt{sk}(\boldsymbol{f}_{i,j})$ built locally at participating sites $j \in \cup_{i=1}^m \mathtt{sites}(\boldsymbol{f}_i)$ (based on the specific join predicates in Q, as detailed in [11]). As shown in the following theorem, simply monitoring the L_2-norm deviations of local-stream sketches is sufficient to guarantee error bounds for the predicted-sketch estimates at the coordinator that are very similar to the corresponding bounds for the simple, centralized case (see Section 2).

Theorem 3.3. *Assume parallel local-stream sketches of size $O(\frac{1}{\epsilon^2}\log(1/\delta))$, and let $\hat{s_i} = \sum_{j \in \mathtt{sites}(\boldsymbol{f}_i)} \mathtt{sk}^p(\boldsymbol{f}_{i,j})$ ($i = 1, \ldots, m$). If each remote site $j \in \mathtt{sites}(\boldsymbol{f}_i)$ satisfies the condition (*), then with probability at least $1 - 2\sum_{i=1}^m k_i\delta$, the predicted-sketch estimate $\Pi_{i=1}^m \hat{s_i}$ at the coordinator lies in the range $\Pi_{i=1}^m \boldsymbol{f}_i \pm (\epsilon + (1+\epsilon)^m ((1+\theta)^m - 1)) \cdot (2^{m-1} - 1)^2 \Pi_{i=1}^m ||\boldsymbol{f}_i|| \approx \Pi_{i=1}^m \boldsymbol{f}_i \pm (\epsilon + m\theta)\,(2^{m-1} - 1)^2 \Pi_{i=1}^m ||\boldsymbol{f}_i||$.*

3.2 Sketch-Prediction Models

We give different options for the sketch-prediction models employed to describe local update behaviors at remote sites. Such models are part of the information exchanged between the remote sites and the coordinator so that both parties are "in-sync" with respect to predicted query results and local-constraint monitoring. If our prediction models result in predicted sketches $\mathtt{sk}^p(\boldsymbol{f}_{i,j})$ that are sufficiently close to the true state of the local sketches at site j, then no communication is required between site j and the coordinator. Thus, it is critical to keep sketch-prediction models concise and, yet, powerful enough to effectively capture *stability* properties in our distributed-tracking environment.[5] In each case, our prediction models consider how the local distribution $\boldsymbol{f}_{i,j}$ changes (as a function of time) between the time of the last communication to the coordinator t_{prev} and the current time t; then, we show how to translate this model to a model for predicting the change in the *sketch* of $\boldsymbol{f}_{i,j}$ over time (Figure 2). As we will see, the linearity properties of sketches play a crucial role in the design of space-, time-, and communication-efficient sketch-prediction models.

Static Model. Our simplest prediction model is the *static model*, which essentially assumes that the local-stream distribution $\boldsymbol{f}_{i,j}$ remains static over time; in other words, our prediction for the distribution $\boldsymbol{f}_{i,j}$ at the current time instant t (denoted by $\boldsymbol{f}^p_{i,j}(t)$) does not change over the time interval $t - t_{prev}$, or $\boldsymbol{f}^p_{i,j}(t) = \boldsymbol{f}_{i,j}(t_{prev})$. This implies that the predicted sketch $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$ employed at both the coordinator and remote site j is exactly the sketch last shipped from site j; that is, $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) = \mathtt{sk}^p(\boldsymbol{f}^p_{i,j}(t)) = \mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev}))$. Such a prediction model is trivial to implement, essentially requiring no additional information to be exchanged between the coordinator and remote sites (besides the sites' local sketches).

Linear-Growth Model. Due to its simplistic nature, the static model can only achieve stability in very "easy" and somewhat unrealistic scenarios, namely when all frequency counts in the $\boldsymbol{f}_{i,j}$ remain reasonably stable. This is clearly not the case, for instance, when local frequency counts are growing as more updates arrive at remote sites. In such cases, a reasonable "strawman" model is to assume that the future of the local distribution will resemble a scaled-up version of its past; that is, assume that $\boldsymbol{f}_{i,j}(t)$ has the same shape as $\boldsymbol{f}_{i,j}(t_{prev})$ with proportionately more elements. Our second, *linear-growth model* is based on this assumption, setting $\boldsymbol{f}^p_{i,j}(t) = \frac{t}{t_{prev}} \boldsymbol{f}_{i,j}(t_{prev})$, i.e., using a linear scaling of $\boldsymbol{f}_{i,j}(t_{prev})$ to predict the current state

[5] A similar notion of prediction models was introduced for the specific problem of tracking one-dimensional quantiles in [6]; instead, we focus on tracking general-purpose randomized sketch summaries of data distributions. Such notions of models are very different from those in [10]: there, models are used in a sensor network to optimize the cost of evaluating one-shot queries by polling specific sensors.

of the distribution. (Scaling by time makes sense, e.g., in a *synchronous-updates* environment, where updates to remote sites arrive regularly at each time tick.) By sketch linearity, this easily implies that the corresponding predicted sketch is simply $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) = \mathtt{sk}(\boldsymbol{f}^p_{i,j}(t)) = \frac{t}{t_{prev}}\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev}))$, a linear scaling of the most recent local sketch of $fv_{i,j}$ shipped to the coordinator (and no additional information need be exchanged between sites and the coordinator).

Velocity/Acceleration Model. Although intuitive, our linear-growth model suffers from at least two important shortcomings. First, it predicts the future behavior of the stream as a linear scaling of the entire history of the distribution, whereas, in many real-life scenarios, only the recent history of the stream may be relevant for such predictions. Second, it imposes a linear, uniform rate of change over the entire frequency distribution vector, and, thus, cannot capture or adapt to shifts and differing rates in the distribution of updates over the vector. Our final, *velocity/acceleration model* addresses these shortcomings by explicitly attempting to build a richer prediction model that uses more parameters to better fit changing data distributions; more specifically, letting $\Delta t = t - t_{prev}$, our velocity/acceleration model predicts the current state of the $\boldsymbol{f}_{i,j}$ distribution as $\boldsymbol{f}^p_{i,j}(t) = \boldsymbol{f}_{i,j}(t_{prev}) + \Delta t \boldsymbol{v}_{i,j} + (\Delta t)^2 \boldsymbol{a}_{i,j}$, where the vectors $\boldsymbol{v}_{i,j}$ and $\boldsymbol{a}_{i,j}$ denote a *velocity* and *acceleration* component (respectively) for the evolution of the $\boldsymbol{f}_{i,j}$ stream. Again, by sketch linearity, this implies the predicted sketch $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) = \mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) + \Delta t\mathtt{sk}(\boldsymbol{v}_{i,j}) + (\Delta t)^2\mathtt{sk}(\boldsymbol{a}_{i,j})$. Thus, to build a predicted sketch at the coordinator under a velocity/acceleration model, we need a velocity sketch $\mathtt{sk}(\boldsymbol{v}_{i,j})$ and an acceleration sketch $\mathtt{sk}(\boldsymbol{a}_{i,j})$. A concrete scheme for computing these two sketches at site j is to maintain a sketch on a window of the W most recent updates to $\boldsymbol{f}_{i,j}$; scaling this sketch by the time difference between the newest and oldest updates stored in the window gives an appropriate velocity sketch to be shipped to the coordinator, whereas the acceleration sketch can be estimated as the difference between the recent and previous velocity sketches scaled by the time difference. In detail, when remote site j detects a violation of its local L_2-norm constraint for $\boldsymbol{f}_{i,j}$ at time t, it computes a new velocity sketch $\mathtt{sk}(\boldsymbol{v}_{i,j})$ based on the window of the W most recent updates to $\boldsymbol{f}_{i,j}$, and estimates a new acceleration sketch $\mathtt{sk}(\boldsymbol{a}_{i,j})$ as the difference between $\mathtt{sk}(\boldsymbol{v}_{i,j})$ and the corresponding velocity sketch at time t_{prev}, scaled by $\frac{1}{t - t_{prev}}$. Note that, the only additional model information that needs to be communicated to the coordinator from site j is the new velocity sketch $\mathtt{sk}(\boldsymbol{v}_{i,j})$ (since the coordinator already has a copy of the previous velocity sketch and so can independently compute the acceleration sketch). Thus, while our richer velocity/acceleration model can give a better fit for dynamic distributions, it also effectively doubles the amount of information exchanged (compared to our simpler prediction models). Furthermore, the effectiveness of our velocity/acceleration predictions can depend on the size of the update window W. While it is possible to set W adaptively for different stream distributions, this problem lies beyond the scope of this paper; instead, we evaluate different settings for W experimentally over real-life data (Section 5).

The following table summarizes the key points for each of our three sketch-prediction models (namely, the model information exchanged between the sites and the coordinator, and the corresponding predicted sketches).

Model	Info.	Predicted Sketch
Static	$\emptyset$	$\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev}))$
Linear-Growth	$\emptyset$	$\frac{t}{t_{prev}}\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev}))$
Velocity/ Acceleration	$\mathtt{sk}(\boldsymbol{v}_{i,j})$	$\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) + \Delta t\mathtt{sk}(\boldsymbol{v}_{i,j})$ $+(\Delta t)^2\mathtt{sk}(\boldsymbol{a}_{i,j})$

Analysis. We analyze the *worst-case* communication cost of our inner-product tracking scheme as a function of the overall approximation error at the coordinator under some simplifying assumptions.

Theorem 3.4. *Assume our static prediction model for an inner-product query $Q(\boldsymbol{f}_1, \boldsymbol{f}_2) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2$ (with ϵ, δ, θ, and k_i as defined earlier), and let $\psi = g_Q(\epsilon, \theta) \approx \epsilon + 2\theta$ denote the error tolerance at the coordinator. Then, for appropriate settings of parameters ϵ and θ (specifically, $\epsilon = \frac{2\psi}{3}$, $\theta = \frac{\psi}{6}$), the worst-case communication cost for a remote site j processing N_j local updates to stream $\boldsymbol{f}_{i,j}$ is $O(\frac{\sqrt{k_i}}{\psi^3}\log(\frac{k_i}{\delta})\log N_j)$.*

That is, assuming that the "distribution factors" k_i of streams in the join query are reasonably small, the worst-case communication cost even for our simplest prediction model is comparable to that of a *one-shot* sketch-based approximate query computation with the same error bounds (Theorem 2.1). (Note, of course, that each counter in the sketches for site j is of size $O(\log N_j)$.) This analysis extends in a natural manner to the case of multi-join aggregates. Providing similar analytical results for our more complex linear-growth and velocity/acceleration models is more complex; instead, we experimentally evaluate different strategies for setting ϵ and θ to minimize worst-case communication over real-life streams in Section 5.

3.3 Time-Efficient Tracking: The Fast-AGMS Sketch

A drawback of AGMS randomized sketches (Section 2) is that every streaming update must "touch" every component of the sketch vector (to update the corresponding randomized linear projection). Since sketch-summary sizes can vary from tens to hundreds of Kilobytes, especially when tight error guarantees are required, e.g., for join or multi-join aggregates [1, 11], touching every counter in such sketches is simply infeasible when dealing with large data rates (e.g., monitoring a high-capacity network link). This problem is compounded in our distributed-tracking scenario where, for each streaming update, a remote site needs to track the difference between a sketch of the updates and an evolving predicted sketch.

Our proposed *Fast-AGMS* sketch structure solves this problem by guaranteeing *logarithmic-time* (i.e., $O(\log(1/\delta)))$ sketch update and tracking costs, while offering essentially the same (in fact, slightly improved) space/accuracy tradeoff as basic AGMS sketches. Our discussion is brief since the structure bears similarities to existing techniques proposed in the context of different (centralized) streaming problems (e.g., [4, 12]), although its application over the bacic AGMS technique for join/multi-join aggregates is novel and requires a different analysis.

A Fast-AGMS sketch for a stream $\boldsymbol{f}$ over $[U]$ (also denoted by $\mathtt{sk}(\boldsymbol{f})$) comprises $b \times d$ counters (i.e., linear projections) arranged in d hash tables, each with b hash buckets. Each hash table $l = 1, \ldots, d$ is associated with (1) a *pairwise-independent hash function* $h_l()$ that maps incoming stream elements uniformly over the b hash buckets (i.e., $h_l : [U] \rightarrow [b]$); and, (2) a family $\{\xi_l[v] : v \in [U]\}$ of four-wise independent $\{-1, +1\}$ random variables (as in basic AGMS). To update $\mathtt{sk}(\boldsymbol{f})$ in response to an insertion/deletion of element v, we use the $h_l()$ hash functions to determine the appropriate buckets in the sketch, setting $\mathtt{sk}(\boldsymbol{f})[h_l(v), l] = \mathtt{sk}(\boldsymbol{f})[h_l(v), l] \pm \xi_l[v]$, for each $l = 1, \ldots, d$. Note that the required time per update is only $O(d)$, since each update touches *only one bucket* per hash table. Now, given two parallel Fast-AGMS sketches $\mathtt{sk}(\boldsymbol{f}_1)$ and $\mathtt{sk}(\boldsymbol{f}_2)$ (using the same hash functions and ξ families), we estimate the inner product $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$ by the sketch "inner product":

$$\mathtt{sk}(\boldsymbol{f}_1) \cdot \mathtt{sk}(\boldsymbol{f}_2) = \underset{l=1,\ldots,d}{\text{median}} \{ \sum_{i=1}^{b} \mathtt{sk}(\boldsymbol{f}_1)[i, l] \cdot \mathtt{sk}(\boldsymbol{f}_2)[i, l] \}.$$

In other words, rather than averaging over independent linear projections built over the entire $[U]$ domain, our Fast-AGMS sketch averages over *partitions* of $[U]$ generated randomly (through the $h_l()$ hash functions). As the following theorem shows, this results in essentially identical space/accuracy tradeoffs as basic AGMS sketching, while requiring only $O(d) = O(\log(1/\delta))$ processing time per update.

Theorem 3.5. *Let $\mathtt{sk}(\boldsymbol{f}_1)$ and $\mathtt{sk}(\boldsymbol{f}_2)$ denote two parallel Fast-AGMS sketches of streams $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$, with parameters $b = O(\frac{1}{\epsilon^2})$ and $d = O(\log(1/\delta))$, where ϵ, $1 - \delta$ denote the desired bounds on error and probabilistic confidence, respectively. Then, with probability at least $1 - \delta$, $||\mathtt{sk}(\boldsymbol{f}_1) - \mathtt{sk}(\boldsymbol{f}_2)||^2 \in (1 \pm \epsilon)||\boldsymbol{f}_1 - \boldsymbol{f}_2||^2$ and $\mathtt{sk}(\boldsymbol{f}_1) \cdot \mathtt{sk}(\boldsymbol{f}_2) \in (\boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \pm \epsilon||\boldsymbol{f}_1||||\boldsymbol{f}_2||)$. The processing time required to maintain each sketch is $O(\log(1/\delta))$ per update.*

Note that tighter error tolerances only increase the size b of each hash table, but not the number of hash tables d (which depends only on the required confidence). Finally, for given ϵ and δ, our Fast-AGMS sketch actually requires *less space* than that of basic AGMS; this is because basic AGMS requires a total of $O(\frac{1}{\epsilon^2} \log(1/\delta))$ hash functions (one for each ξ family), whereas our Fast-AGMS sketch only needs a pair of hash functions per hash table for a total of only $O(\log(1/\delta))$ hash functions.

In our solution, each update to the local $\boldsymbol{f}_{i,j}$ at site j requires checking the local sketch-tracking condition on the L_2 norm of the divergence of the site's true sketch from the corresponding predicted sketch. Implementing such a sketch-tracking scheme directly over local sketches of size $O(\frac{1}{\epsilon^2} \log(1/\delta))$ would imply a time complexity of $O(\frac{1}{\epsilon^2} \log(1/\delta))$ per update (to recompute the required norms) — this complexity can easily become prohibitive when dealing with rapid-rate update streams and tight error-bound requirements. Fortunately, as the following theorem demonstrates, we can reduce the sketch-tracking overhed in only $O(\log(1/\delta))$ per update by computing the tracking condition in an *incremental* fashion over the input stream. Our tracking algorithm makes crucial use of the Fast-AGMS sketch structure, as well as concise ($O(\log(1/\delta))$-size) precomputed data structures to enable incremental sketch tracking. We focus primarily on our most general velocity/acceleration model, since both the static and linear-growth models can be thought of as instances of the velocity/acceleration model with certain parameters fixed.

Theorem 3.6. *Assuming Fast-AGMS sketches of size $O(\frac{1}{\epsilon^2} \log(1/\delta))$, the computation of the sketch tracking condition (*) at site j can be implemented in $O(\log(1/\delta))$ time per update, where the predicted sketch $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$ is computed in the velocity/acceleration model.*

If our tracking scheme detects that a θ bound has been violated, we must recompute the parameters of the sketch-prediction model and send sketch information to the coordinator. Such communications necessarily require $O(\frac{1}{\epsilon^2} \log(1/\delta))$ time, but occur relatively rarely.

3.4 Handling Other Query Classes

We outline how our results apply to the other query classes introduced in Section 2. The basic intuition is that such queries can be viewed as special inner products of the distribution (e.g., with wavelet-basis vectors [14]), for which sketches can provide guaranteed-quality estimates. The predicted sketch of $\boldsymbol{f}_i$ at the coordinator can be treated as a $g(\epsilon, \theta)$-approximate sketch of $\boldsymbol{f}_i$, which accounts for both sketching error (ϵ) and remote-site deviations (θ).

- *Range Queries, Point Queries, and Heavy Hitters.* A given range query $R(\boldsymbol{f}_i, a, b)$ can be reposed as an inner product with a vector $\boldsymbol{e}_{[a,b]}$ where $\boldsymbol{e}_{[a,b]}[v] = 1$ if $a \leq v \leq b$, and 0 otherwise. This implies the following theorem.

Theorem 3.7. *Assume local-stream sketches of size $O(\frac{1}{\epsilon^2} \log(1/\delta))$ and let $\hat{s}_i = \sum_{j \in \mathtt{sites}(\boldsymbol{f}_i)} \mathtt{sk}^p(\boldsymbol{f}_{i,j})$. If for each remote site $j \in \mathtt{sites}(\boldsymbol{f}_i)$ satisfies the condition (*), then with probability at least $1 - k_i\delta$, $\hat{s}_i \cdot \mathtt{sk}(\boldsymbol{e}_{[a,b]}) \in R(\boldsymbol{f}_i, a, b) \pm \epsilon + (1 + \epsilon)^2((1 + \theta)^2 - 1))(b - a + 1)||\boldsymbol{f}_i||$.*

An immediate corollary is that point queries can be answered with $\approx (\epsilon + 2\theta)||\boldsymbol{f}_i||$ error. Heavy-hitter queries can be answered by asking all U_i point queries, and returning those v whose estimate exceeds $\phi R(\boldsymbol{f}, a, b)$ (with guarantees similar to the centralized, one-shot case [4]).

- *Histogram and Wavelet Representations.* Gilbert et al. [14] demonstrate how to use ϵ-approximate sketches to find B-term Haar-wavelet transforms that carry at least $1-\epsilon$ of the energy of best B-term representation if this representation has large coefficients. In our setting, the sketch at the coordinator is essentially $g(\epsilon, \theta)$-approximate sketch; thus, our analysis in conjunction with Theorem 3 of [14], imply that our schemes can track a $1 - g(\epsilon, \theta)$ approximation to the best B-term wavelet representation at the coordinator. Similarly, Thaper et al. [22] show how to use ϵ-approximate sketches to find an approximate histogram representation with error at most $1 + B\epsilon$ times the error of the best B-bucket multi-dimensional histogram. Combining our results with Theorem 3 of [22], we have a scheme for tracking a $1 + Bg(\epsilon, \theta)$ approximation to the best B-bucket multi-dimensional histogram.

4 Extensions

We consider modifications to accommodate answering queries based only on recent updates, and incorporating different query models.

Sliding Windows and Exponential Decay. In the sliding window case, the current distribution $\boldsymbol{f}_i$ is limited to only those updates occurring within the last t_w time units, for some fixed value of t_w. We modify the tracking condition: the remote sites build a sketch of the most recent t_w time units, and track whether a predicted sketch for this interval is within θ error of the interval norm. The role of the coordinator remains the same: to answer a query, it uses the predicted sketch, as above. In the case that the site is not space-constrained, the remote site can buffer the updates that occurred in the window. When the oldest update v in the buffer is more than t_w time units old, it can be treated as an update $< i, v, -1 >$ to $\boldsymbol{f}_i$. The effect of the original update of v is subtracted from the sketch, and so the sketch only summarizes those updates within the window of t_w. Using the above efficient tracking method, the asymptotic cost is not altered in the amortized sense, since each update is added and later subtracted once, giving an amortized cost of $O(\log(1/\delta))$ per update.

The exponential decay model is a popular alternative to the sliding window model (see, e.g., [14]). Briefly, the current distribution $\boldsymbol{f}_i(t)$ is computed as $\boldsymbol{f}_i(t) = \lambda^{t-t'} \boldsymbol{f}_i(t')$ for a positive decay constant $\lambda < 1$ — for example, $\lambda = 0.95$ or 0.99. Updates are processed as before, so an update v means $\boldsymbol{f}_i(t)[v] \leftarrow \boldsymbol{f}_i(t)[v] + 1$. As in the sliding window case, the action at the coordinator is unchanged: given a suitable model of how the (exponentially-decayed) distribution changes, the coordinator uses the predicted sketch to answer queries. At the remote site, the tracking condition is again checked. Since the decay operation is a linear transform of the input, the sketch of the decayed distribution can be computed by decaying the sketch: $\mathsf{sk}(\boldsymbol{f}_i(t)) = \lambda^{t-t'} \mathsf{sk}(\boldsymbol{f}_i(t'))$ Applying this directly would mean the tracking operation takes time $O(\frac{1}{\epsilon^2} \log(1/\delta))$, but by devoting some extra space to the problem, we can track the condition in time $O(\log(1/\delta))$ again. In summary,

Theorem 4.1. *The sketch tracking condition (*) can be tracked in time $O(\log(1/\delta))$ per update in both the sliding window and the exponential decay streaming models.*

Alternate Sketch-Prediction Models. We outlined three distinct approaches to sketch prediction, each building progressively richer models to attempt to capture the behavior of local stream distributions over time. Our most sophisticated model explicitly tries to model both first-order (i.e., "velocity") and second-order (i.e., "acceleration") effects in the local update-stream rates while increasing the amount of sketching information communicated to the coordinator by a factor of only two. One can envisage other models of evolving local distributions, and translating these into predicted sketches by applying the linearity properties of the sketch transformation. Other variations are also possible. Thus far, our models operate on whole sketches at a time; it is possible, however, to design "finer-grained" models that consider different parts of the distribution separately. For instance, individual data elements with high counts in the $\boldsymbol{f}_{i,j}$ distribution carry the highest impact on the norm of the distribution. Thus, we can separate such "heavy-hitter" elements from the rest of the distribution and model their movements separately (e.g., tracking an acceleration model), while using a sketch only for tracking the remainder of the distribution. Once a local constraint is violated, then it may be possible to restore the constraint by only shipping information on some of the heavy-hitter items, instead of shipping an entire sketch – clearly, this may drastically reduce the amount of communication required. At a high level, this approach is similar to the idea of "skimming sketches" of Ganguly et al. [12], but for the purpose of decreasing communication rather than increasing accuracy. We will explore such sketch-skimming approaches in the full version of this work.

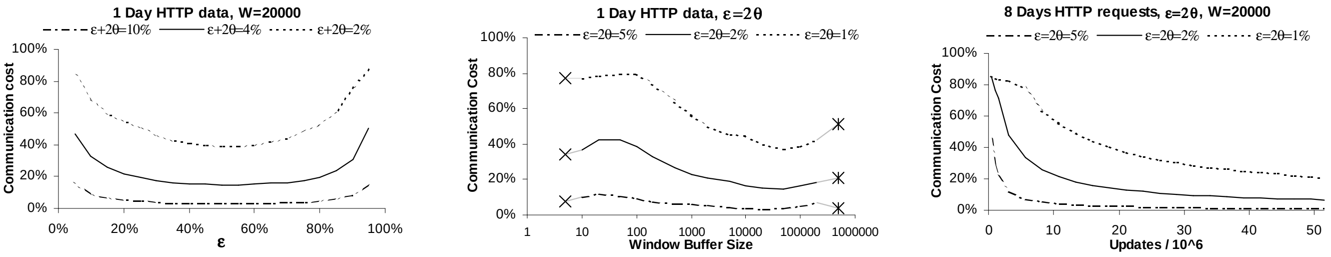
5 Experimental Study

5.1 Testbed and Methodology

We implemented a test system that simulated running our protocols in C. [6] Experiments were run on a single machine, simulating the actions of each of k sites and the coordinator. For each experimental simulation, all remote sites used the same class of prediction model with the same tracking parameters ϵ, θ.

We report the results of experiments run on data sets drawn from the Internet Traffic Archive [17], representing HTTP requests sent to servers hosting the World Cup 1998 Web Site. Servers were hosted in four geographic locations. Therefore, we modeled this system with four remote sites, one handling requests to each location. We tracked the relations defined by this sequence of requests, using the

[6] Throughout, we set the probability of failure, $\delta = 1\%$.



(a) **(b)** **(c)**

Figure 4: Experiments on real data: (a) Tradeoff between the parameters θ and ϵ. (b) Effect of varying the window size used to estimate the "velocity" sketch. (c) Communication cost as number of updates increase.

"objectID" attribute as the attribute of interest. This seems a good approximation of many typical data sets, taking on a large number of possible values with a non-uniform distribution. We obtained similar results to those reported here when using different data sets and settings.

Throughout, we measure the *communication cost*, as the ratio between the total communication used by a protocol (in bytes) divided by the total cost to send every update in full (in bytes). For example, if our protocol sent 3 sketches, each of which was 10KB in size, to summarize a set of 50,000 updates, each of which can be represented as a 32bit integer, then we compute the communication cost as 15%. Our goal is to drive this cost as low as possible. When measuring the accuracy of our methods, we compute an estimated result est, and (for testing) compute the exact answer, $true$. The error is then given by $\frac{|true-est|}{true}$, which gives a fraction, 0% being perfect accuracy.

5.2 Experimental Results

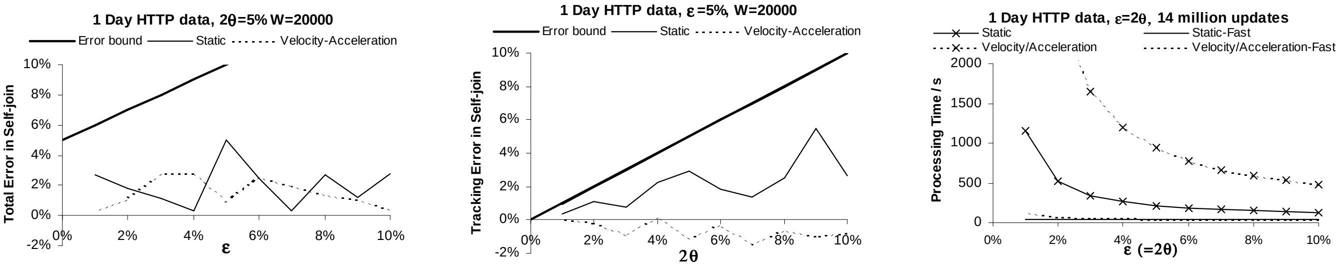
Setting Parameters and Tradeoffs. We investigated the tradeoff between parameters ϵ and θ in order to guarantee a given global error bound, and the setting of the parameter W for the velocity/acceleration model. We took one day of HTTP requests from the World Cup data set, which yielded a total of 14 million requests. Figure 4 (a) shows the effect of varying ϵ and θ subject to $\epsilon+2\theta=\psi$, for $\psi = 10\%$, 4%, and 2% error rate. In each case, we verified that the total error was indeed less than ψ. The communication cost is minimized for ϵ roughly equal to 0.55ψ. Our analysis in Section 3.2 showed that for a worst case distribution under the static model, ϵ should be around 0.66ψ. In practice, it seems that a slightly different balance gives the lowest cost, although the trade-off curve is very flat-bottomed, and setting ϵ between 0.3ψ and 0.7ψ gives similar bounds. We have shown the curves for the velocity/acceleration model with $W = 20000$; curves for the different models and different settings of W look similar. For the remainder of our experiments, we set $\epsilon = 0.5\psi$ and $\theta = 0.25\psi$, giving $g(\epsilon, \theta) \approx \psi$.

In Figure 4 (b), we show the effect of varying the window size W for the velocity/acceleration model on the communication cost for three values of $\psi = \epsilon + 2\theta$. In order to show all three models on the same graph, we have shown the static model cost as the leftmost point (plotted with a cross), since this can be thought of as the velocity/acceleration model with no history used to predict velocity. Similarly, we plot the cost of the linear growth model as the rightmost point on each curve (marked with an asterisk), since this can be thought of as using the whole history to predict velocity. We see that for the best setting of the window size the velocity/acceleration model outperforms both the other models by at least a third, but it is sensitive to the setting of W: too small or too large, and the overall communication cost is noticeably worse than the best value. The static model gets close to the worst cost, while the linear growth model does quite well, but still about a third more than the best velocity/acceleration model. For this data set, irrespective of the $g(\epsilon, \theta)$ value the best setting of W is in the range 10000–100000. Therefore, for the remainder of our experiments, we focus on the velocity/acceleration model with $W = 20000$.

Communication cost. We look at how the communication cost evolves with time in Figure 4 (c), using the velocity/acceleration model. This experiment was performed on a larger data set from a week of HTTP requests to the World Cup data sets, totaling over 50 million updates. We see that the cost is initially high, as the remote site adapts to the stream, but as the number of updates increases, then the requirement for communications drops. For the higher error bounds, there are long periods of stability.

Accuracy of Approximate Query Answers. Our first set of experiments focused on the communication cost of our proposed protocols. We now consider the accuracy they provide for answering queries at the coordinator, and the time cost at the remote sites. In Figure 5 (a), we plot the error in answering queries at the coordinator based on processing the one day of data from the World Cup data set. Here, we have fixed θ, and plotted the observed accuracy for computing the size of a self-join as ϵ varies when we have processed all updates. We show with a heavy line the worst case error bound $\epsilon + 2\theta \approx g(\epsilon, \theta)$.

In Figure 5 (b), we attempt to separate the sketch error from the tracking error, by computing the approximation we would get if the remote site sent the sketch of its current distribution to the coordinator when the self-join query was posed. In this figure, we have subtracted this error from the



(a) **(b)** **(c)**

Figure 5: Quality experimenrs: (a) Estimation quality, fixed θ, varying ϵ. (b) Estimation quality due to tracking delay with sketch error subtracted, as θ varies. (c) Timing cost, comparing fast tracking methods to performing sketch estimation every step.

total error to give an indication of how much error is due to tracking as θ varies. The negative values seen in the results for the velocity/acceleration model indicate that the answer given by using this prediction model at the coordinator is actually *more* accurate than if the coordinator requested each site to send it a sketch at query time. This shows an unexpected benefit. Our worst-case bounds must assume that the errors from sketching and tracking are additive, but, in some cases, these errors can partially *cancel out*. For the static case, we more clearly see the trend for the tracking error to decrease as θ decreases to zero.

Timing Results. We compared the implementation of our methods using Fast-AGMS sketches and our fast sketch-tracking scheme against the same prediction models implemented with a naive tracking method with time complexity linear in the sketch size (Figure 5 (c)). Communication cost and accuracy of both versions was the same in all cases. For small ϵ, the fast velocity/acceleration method is more expensive since, while update operations are still fast, recomputing the sketches when tracking bounds are broken contributes more significantly to the overall cost. For $\epsilon \geq 3\%$, the cost was 36 seconds in the static case, and 50 seconds for the velocity/acceleration model to process all 14 million updates — an average overhead of 3 microseconds per update on our experimental setup (2.4GHz Pentium desktop).

Experimental Conclusions. We saw that significant communication savings are possible: with an approximation factor of 10%, the communication cost can be less than 3% of sending every update. Time overhead is minimal, a few microseconds to update the necessary tracking structures, and typically a few kilobytes per sketch. The velocity/acceleration model gives best performance, if enough information about the streams is known to choose a good setting of the window parameter W; else linear growth provides adequate results, and requires no extra parameters.

6 Conclusions

We have presented novel algorithms for tracking complex queries over multiple streams in a general distributed setting. Our schemes are optimized for tracking high-speed streams, and result in very low processing and communication costs, and significant savings over naive updating schemes. Our key results show that any query that can be answered using sketches in the centralized model can be tracked efficiently in the distributed model, with low space, time, and communication.

References

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". *ACM PODS*, 1999.

[2] N. Alon, Y. Matias, and M, Szegedy. "The Space Complexity of Approximating the Frequency Moments". *ACM STOC*, 1996.

[3] B. Babcock and C. Olston. "Distributed Top-K Monitoring". *ACM SIGMOD*, 2003.

[4] M. Charikar, K. Chen, and M. Farach-Colton. "Finding Frequent Items in Data Streams". *ICALP*, 2002.

[5] G. Cormode and S. Muthukrishnan. "An improved data stream summary: The count-min sketch and its applications". *LATIN*, 2004.

[6] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles". *ACM SIGMOD*, 2005.

[7] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications". *ACM SIGMOD*, 2003.

[8] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. "Distributed Set-Expression Cardinality Estimation". *VLDB*, 2004.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining Stream Statistics over Sliding Windows". *ACM-SIAM SODA*, 2002.

[10] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. "Model-Driven Data Acquisition in Sensor Networks". *VLDB*, 2004.

[11] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing Complex Aggregate Queries over Data Streams". *ACM SIGMOD*, 2002.

[12] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Data-Stream Join Aggregates Using Skimmed Sketches". *EDBT*, 2004.

[13] P. B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports". *VLDB*, 2001.

[14] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". *VLDB*, 2001.

[15] M. B. Greenwald and S. Khanna. "Space-Efficient Online Computation of Quantile Summaries". *ACM SIGMOD*, 2001.

[16] M. B. Greenwald and S. Khanna. "Power-Conserving Computation of Order-Statistics over Sensor Networks". *ACM PODS*, 2004.

[17] Internet traffic archive. (`http://ita.ee.lbl.gov/`).

[18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "The Design of an Acquisitional Query Processor for Sensor Networks". *ACM SIGMOD*, 2003.

[19] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. "Finding (Recently) Frequent Items in Distributed Data Streams". *IEEE ICDE*, 2005.

[20] G. Singh Manku and R. Motwani. "Approximate Frequency Counts over Data Streams". *VLDB*, 2002.

[21] C. Olston, J. Jiang, and J. Widom. "Adaptive Filters for Continuous Queries over Distributed Data Streams". *ACM SIGMOD*, 2003.

[22] N. Thaper, S. Guha, P. Indyk, and N. Koudas. "Dynamic Multidimensional Histograms". *ACM SIGMOD*, 2002.