Approximate Continuous Querying over Distributed Streams

GRAHAM CORMODE
AT&T Labs–Research
and
MINOS GAROFALAKIS
Yahoo! Research & University of California, Berkeley

While traditional database systems optimize for performance on one-shot query processing, emerging large-scale monitoring applications require continuous tracking of complex data-analysis queries over collections of physically distributed streams. Thus, 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. In a nutshell, our algorithms 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 end 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. Experiments with real data validate our approach, revealing significant savings over naive solutions as well as our analytical worst-case guarantees.

Categories and Subject Descriptors: H.4.3 [**Information Systems Applications**]: Communications Applications; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Continuous distributed monitoring, data stream algorithms, data synopses, approximate query processing

Parts of this work were done while G. Cormode was with Bell Labs, Alcatel-Lucent Technologies, and M. Garofalakis was with Bell Labs, Alcatel-Lucent Technologies, and Intel Research Berkeley. This paper extends work originally published as [Cormode and Garofalakis 2005].
Authors' current addresses: G. Cormode AT&T. Labs-Research, 180 Park Ave., Florham Park, NJ; email: cormode@research.att.com; M. Garofalakis, Yahoo! Research, 2821 Mission College Blvd., Santa Clara, CA; email: minos@yahoo-inc.com.

ACM Reference Format:
Cormode, G. and Garofalakis, M. 2008. Approximate continuous querying over distributed streams. ACM Trans. Datab. Syst. 33, 2, Article 9 (June 2008), 39 pages. DOI = 10.1145/1366102.1366106 http://doi.acm.org/10.1145/1366102.1366106

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 done 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 are typically impressive computing facilities, monitoring 100's of routers, 1000's of links and interfaces, and blisteringly-fast sets of events at different layers of the network infrastructure (ranging from fiber-cable utilizations to packet forwarding at routers, to VPNs and higher-level transport constructs). The NOC has to continuously track and correlate usage information from a multitude of monitoring points in order to quickly detect and react to hot spots and floods, failures of links or protocols, intrusions, and attacks. A different class of applications is one in which monitoring is the goal in itself. For instance, consider a wireless network of seismic, acoustic, and physiological sensors that are deployed for habitat, environmental, and health monitoring. 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. Similar issues arise in sophisticated satellite-based systems that do atmospheric monitoring for weather patterns.

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 [Cranor et al. 2003]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [Madden et al. 2003]). 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. 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 [Greenwald and Khanna 2001], estimating distinct values [Gibbons 2001], and set-expression cardinalities [Ganguly et al. 2003], counting frequent elements (i.e., "heavy hitters") [Charikar et al. 2002; Cormode and Muthukrishnan 2003; Manku and Motwani 2002], approximating large Haar-wavelet coefficients [Gilbert et al. 2001], and estimating join sizes and stream norms [Alon et al. 1999; Alon et al. 1996; Dobra et al. 2002]. As already mentioned, all these 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 [Greenwald and Khanna 2004] and heavy hitters [Manjhi et al. 2005]; 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.

It is important to realize that each of the dimensions of our problem (distributed, continuous, and space-constrained) induce specific technical bottlenecks.

For instance, even efficient streaming solutions at individual sites can lead to constant updates on the distributed network and become highly communication-inefficient when they are directly used in distributed monitoring. Likewise, morphing one-shot solutions to continuous problems entails propagating each change and recomputing the solutions which is communication inefficient, or involves periodic updates and other heuristics that can no longer provide real-time estimation guarantees.

Prior research has looked at the monitoring of *single values*, and building appropriate models and filters to avoid propagating updates if these are insignificant compared to the value of a simple aggregate (e.g., to the SUM of the distributed values). Olston et al. [2003] propose a scheme based on "adaptive filters"—that is, bounds around the value of distributed variables, which shrink or grow in response to relative stability or variability, while ensuring that the total uncertainty in the bounds is at most a user-specified bound δ. [Jain et al. 2004] propose building a Kalman Filter for individual values, and only propagating an update in a value if it falls more than δ away from the predicted value. The BBQ system [Deshpande et al. 2004] builds a dynamic, multidimensional probabilistic model of a set of distributed sensor values (viewed as random variables) to drive acquisitional query processing. Given a simple SQL-style query, the system determines whether it is possible to answer the query only from the model information, or whether it is necessary to poll certain locations for up-to-date information. This was extended to the continuous case in the Ken system [Chu et al. 2006], which ensured that the probabilistic model at the central site was kept up to date, to reflect changes in the distributions at remote sites. The approach involved finding a compromise between the fully-independent approach of Kalman filters, and the fully-correlated approach of the BBQ system, and instead capturing correlations only between small cliques of random variables. A common aspect of all these earlier works is that they typically consider only a small number of monitored values per site, and assume that it is feasible to locally monitor and/or build a model for each such value. In contrast, our problem setup is much more complex, as each resource-limited site monitors a *streaming distribution of a large number of values* and cannot afford to explicitly capture or model each value separately. Moreover, for the class of complex aggregate queries that we study (e.g., join or multi-join size), no prior work offers any guaranteed bound on the quality of the query answer, or any guaranteed trade-off between accuracy and communication cost. While one could conceivably partition an overall aggregate error bound (e.g., on the join size) to error bounds for individual values, such a mapping is nontrivial for our class of aggregates; furthermore, such per-value bounds are likely to be much too stringent, thus resulting in excessive communication.

Closest in spirit to our work are the results of Babcock and Olston [2003] and Das et al. [2004], as well as our work on distributed quantile tracking [Cormode et al. 2005]. All these efforts explicitly 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, Babcock and Olston [2003] consider tracking approximate top-k values over dynamically-changing numeric values spread over multiple sources,

whereas Das et al. [2004] discuss monitoring of approximate set-expression cardinalities over physically distributed element streams. Similarly, our recent work [Cormode et al. 2005] 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 *multidimensional* data summaries).

For instance, Das et al. [2004] employ ideas similar to the adaptive filter bounds of Olston et al. [2003] for the distributed monitoring of set-expression cardinalities; since their estimation problem relies on set semantics, they propose a scheme for effectively *charging* local changes against a site's error tolerance. Similarly, Babcock and Olston [2003] focus on monitoring the top-k (i.e., k most frequent) values over remote data streams; their techniques ensure the validity of the current top-k set (at the coordinator) by installing appropriate arithmetic constraints at each site. Once again, these earlier papers focus on specific distributed-monitoring queries (namely, simple aggregates, set-expression cardinalities, and top-k), and are not always applicable to more general settings (specifically, for monitoring summaries of the entire data distribution or more complex, *holistic aggregates* over the remote sites).

Following our original conference paper [Cormode and Garofalakis 2005], Sharfman et al. [2006] have proposed an approach for efficiently monitoring the value of a general function over distributed data relative to a given threshold. Their solution relies on interesting geometric arguments for breaking up a global threshold condition on a function into "safe" local conditions that can be checked locally at each site. The primary focus of Sharfman et al. [2006] is on a slightly different problem, namely, monitoring a *distributed trigger* condition over physically distributed data, and not continuously monitoring a distributed query result with approximation-error guarantees. Still, some of their geometric techniques could be applicable in our setting, and combining their ideas with the results in this paper is an interesting avenue for future work in this area. Other recent work on distributed-trigger monitoring includes Keralapura et al. [2006]; Huang et al. [2007a, 2007b].

Our Contributions. In this article, 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 Distributed 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 (1) different data-streaming models that place more emphasis on recent updates to the stream (using either *sliding-window* or *exponential-decay* mechanisms); and (2) more complex, hierarchical-monitoring architectures, where the communication network is arranged as a *tree-structured* hierarchy of nodes (such as a sensornet *routing tree* [Madden et al. 2003]).

—*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.

2. PRELIMINARIES

In this section, we describe the key elements of our distributed stream-processing architecture and define the class of distributed query-tracking problems addressed in this article. We also provide some necessary background material on randomized stream-sketching techniques that lie at the core of our proposed solutions.

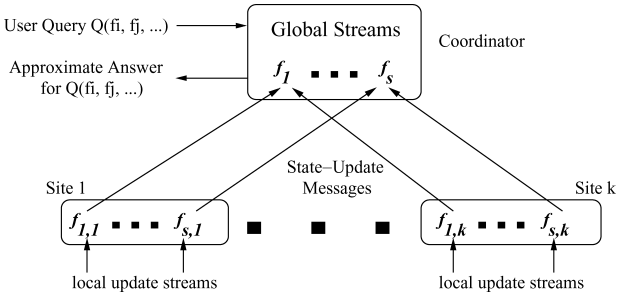


Fig. 1. Distributed stream processing architecture.

2.1 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 [Babcock and Olston 2003; Cormode et al. 2005; Das et al. 2004; Olston et al. 2003], 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, multi-sets) $\boldsymbol{f}_{1,j}, \ldots, \boldsymbol{f}_{s,j}$ 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 ith update stream at site j as well as the underlying element multi-set/frequency distribution in what follows.) Each stream update at remote site j is a triple of the form $\langle i, v, \pm 1 \rangle$, 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

Table I. Summary of Main Notation

Symbol	Description
k	Number of distributed monitoring sites
s	Number of monitored frequency distributions
f	Frequency vector defining a distribution
$\texttt{sites}(f)$	Distributed sites possessing components of $\boldsymbol{f}$
k_i	Shorthand for $\lvert\texttt{sites}(\boldsymbol{f}_i)\rvert$
U	High-dimensional universe (domain) over which frequency vectors are defined
t	(Current) timestamp
t_{prev}	Timestamp of most recent update from a given site
Δ	Shorthand for $t - t_{prev}$
$\boldsymbol{v}, \boldsymbol{a}$	Velocity and acceleration vectors
$\texttt{sk}(f)$	Compact sketch of a distribution $\boldsymbol{f}$
$\texttt{sk}^p(f)$	*Predicted* sketch of $\boldsymbol{f}$ which can be computed from simple prediction model
ξ	Hash function used to define sketches
ϵ	Fractional error desired
δ	Probability of returning a result outside approximation bounds
θ	Bound on local deviation of L_2 norm from prediction
$g(\epsilon, \theta)$	Overall error as a function of ϵ and θ

distributed system). The unqualified notation $\boldsymbol{f}_{i,j}$ typically refers to the *current* state of the frequency vector. Table I summarizes some of the key notational conventions used in this paper; additional notation is introduced when necessary. Detailed symbol definitions are provided at the appropriate locations in the text.

Note that handling delete operations substantially enriches our distributed streaming model; for instance, it allows us to effectively handle tracking over *sliding windows* of the streams by simply issuing implicit delete operations for expired stream items no longer in the window of interest at remote sites. We also discuss the extension of our techniques to more complex distributed-tracking architecture, where the underlying communication network is structured as a *multilevel tree hierarchy* (such as the routing trees typically built over sensornet deployments [Madden et al. 2003]).

2.2 Problem Formulation

For each $i \in \{1, \ldots, s\}$, we define the *global* frequency distribution vector $\boldsymbol{f}_i$ for the ith 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 substreams 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 $k_i = \lvert\texttt{sites}(\boldsymbol{f}_i)\rvert$ (hence $k_i \leq k$). 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 generally, *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 $\boldsymbol{f}_i \bowtie \boldsymbol{f}_l \bowtie \boldsymbol{f}_m \cdots$ [Dobra et al. 2002]; 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 Dobra et al. [2002]).

—*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 Alon et al. [1996].

—*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 $\phi < 1$ [Charikar et al. 2002; Cormode and Muthukrishnan 2004].

—*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 $\boldsymbol{f}_i$ [Gilbert et al. 2001; Thaper et al. 2002].

In general, any problem which requires an accurate estimation of the L_2 norm of a vector or vectors, or which requires accurate estimation of vector inner-products, can be continuously monitored on top of the framework we propose here. However, this does not include several other queries, such as tracking count-distinct or clusterings of data, which require other approaches in order to monitor accurately.

Approximate Query Answering. 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) [Deshpande et al. 2004; Madden et al. 2003]. 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.

2.3 Randomized Sketching of Streams

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 [Alon et al. 1996; Alon et al. 1999; Cormode and Muthukrishnan 2004; Gilbert et al. 2001; Dobra et al. 2002]. 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 [Alon et al. 1996; Alon et al. 1999]:[1] The ith entry in an AGMS sketch $\mathtt{sk}(\boldsymbol{f})$ is defined as the

[1]Our techniques and results can also be extended to other randomized stream sketching methods, such as the *Count-Min sketches* [Cormode and Muthukrishnan 2004]; the details are quite straightforward, and are omitted in order to simplify the exposition.

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 on-line in small (i.e., $O(\log U)$) space [Alon et al. 1996]. 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

$$\mathtt{sk}(\boldsymbol{f})[i] = \mathtt{sk}(\boldsymbol{f})[i] + \xi_i[v] \qquad \text{(respectively, } \mathtt{sk}(\boldsymbol{f})[i] = \mathtt{sk}(\boldsymbol{f})[i] - \xi_i[v])$$

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 $|x - y| \leq |z|$.) For these sketches, we use the standard "inner product" operator over sketch vectors as shorthand for a slightly more complex operator, involving both averaging and median-selection operations over the sketch-vector components [Alon et al. 1999; Alon et al. 1996]—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

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

THEOREM 2.1 ([ALON ET AL. 1999; ALON ET AL. 1996]). *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 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* $O(\frac{1}{\epsilon^2}\log(1/\delta))$ *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 $\epsilon\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|$. 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 $\epsilon' = \epsilon(\boldsymbol{f}_1 \cdot \boldsymbol{f}_2)/(\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|)$, 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 [Alon et al. 1999].

The results in Theorem 2.1 can be extended in a natural manner to the case of multi-join aggregate queries [Dobra et al. 2002]: 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 [Dobra et al. 2002]), the inner product of the sketches $\Pi_{i=1}^{m}\mathtt{sk}(\boldsymbol{f}_i)$ (which is once again defined using median-selection and averaging over terms of the form $\Pi_{i=1}^{m}\mathtt{sk}(\boldsymbol{f}_i)[i, j]$) 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 Dobra et al. [2002].

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 [Babcock and Olston 2003; Das et al. 2004]) 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, remote 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 a high-probability error guarantee on the quality of the approximate query answers returned to users.

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 $\{\boldsymbol{f}_i : i = 1, \ldots, s\}$.

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 analyze the communication cost of these schemes. This 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.[2] 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 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 as 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.

In the remainder of this section, we discuss the details of our distributed query-tracking schemes, and our proposed sketch-prediction models

[2]To simplify the exposition, we assume that communications with the coordinator are instantaneous. In the case of nontrivial delays in the underlying communication network, techniques based on time-stamping and message serialization can be employed to ensure correctness, as in Olston et al. [2003]. Thus delays only impact the time to respond to events. Likewise, we do not explicitly treat loss of messages, and instead assume that the low-level network communications ensure acknowledgement of messages.

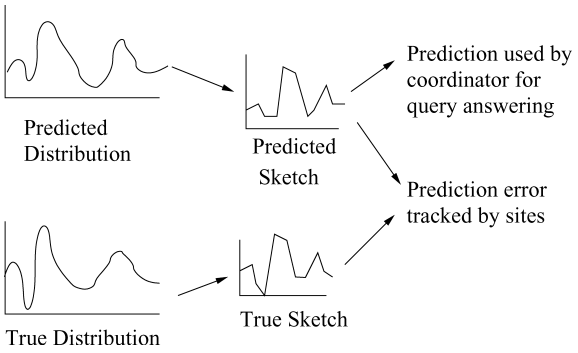


Fig. 2. Schematic of sketch-prediction-based tracking.

for capturing remote-site behavior. In addition, we introduce a 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 substreams $\boldsymbol{f}_{1,j}$ 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 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.[3]

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*

[3]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 ϵ, δ, θ;
"distribution factor" k_i for stream i.
1. UpdateSketch($\mathtt{sk}(\boldsymbol{f}_{i,j}), \langle i, v, \pm 1\rangle$) *//update current and*
2. UpdatePredictedSketch($\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$) *//predicted sketches*
3. **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**
4. Compute sketch-prediction model $\mathtt{predModel}(\boldsymbol{f}_{i,j})$
5. 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 $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$.
1. **for** $i := 1$ **to** 2 **do**
2. Set $\mathtt{sk}^p(\boldsymbol{f}_i(t)) := 0$
3. **for each** $j \in \mathtt{sites}(\boldsymbol{f}_i)$ **do**
4. $\mathtt{sk}^p(\boldsymbol{f}_i(t)) := \mathtt{sk}^p(\boldsymbol{f}_i(t)) + \mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$
5. **return** $\mathtt{sk}^p(\boldsymbol{f}_1(t)) \cdot \mathtt{sk}^p(\boldsymbol{f}_2(t))$

Fig. 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).

$\mathtt{sk}^p(\boldsymbol{f}_{i,j})$ 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 $j \in \mathtt{sites}(\boldsymbol{f}_i)$ (where $i \in \{1, 2\}$). After shipping its local sketch $\mathtt{sk}(\boldsymbol{f}_{i,j})$ 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 $\mathtt{sk}(\boldsymbol{f}_{i,j}(t))$ from the corresponding predicted sketch $\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))$ employed for estimation at the coordinator. The site checks the following condition at every time instant t:

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

that is, a communication to the coordinator is triggered only if the relative L_2-norm deviation of the local, up-to-date sketch $\mathtt{sk}(\boldsymbol{f}_{i,j}(t))$ from the corresponding predicted sketch exceeds $\frac{\theta}{\sqrt{k_i}}$ (recall, $k_i = |\mathtt{sites}(\boldsymbol{f}_i)|$). 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 $\hat{s}_i = \sum_{j \in \mathtt{sites}(\boldsymbol{f}_i)} \mathtt{sk}^p(\boldsymbol{f}_{i,j})$ $(i \in \{1, 2\})$. Also, assume that, for each remote site $j \in \mathtt{sites}(\boldsymbol{f}_i)$ $(i \in \{1, 2\})$, the condition (*) is satisfied. Then, with probability at least $1 - 2(k_1 + k_2)\delta$,*

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

PROOF. Consider the inner product of the "global" predicted sketches $\hat{s}_1 \cdot \hat{s}_2$—algebraic manipulation gives:

$$\begin{aligned}
\hat{s}_1 \cdot \hat{s}_2 &= \left(\sum_j \mathtt{sk}^p(\boldsymbol{f}_{1,j})\right) \cdot \left(\sum_j \mathtt{sk}^p(\boldsymbol{f}_{2,j})\right) \\
&= \left(\sum_j ((\mathtt{sk}^p(\boldsymbol{f}_{1,j}) - \mathtt{sk}(\boldsymbol{f}_{1,j})) + \mathtt{sk}(\boldsymbol{f}_{1,j})\right) \\
&\quad \cdot \left(\sum_j ((\mathtt{sk}^p(\boldsymbol{f}_{2,j}) - \mathtt{sk}(\boldsymbol{f}_{2,j})) + \mathtt{sk}(\boldsymbol{f}_{2,j})\right) \\
&= \sum_j \mathtt{sk}(\boldsymbol{f}_{1,j}) \cdot \sum_j \mathtt{sk}(\boldsymbol{f}_{2,j}) + \sum_j (\mathtt{sk}^p(\boldsymbol{f}_{1,j}) - \mathtt{sk}(\boldsymbol{f}_{1,j})) \cdot \sum_j \mathtt{sk}(\boldsymbol{f}_{2,j}) \\
&\quad + \sum_j (\mathtt{sk}^p(\boldsymbol{f}_{2,j}) - \mathtt{sk}(\boldsymbol{f}_{2,j})) \cdot \sum_j \mathtt{sk}(\boldsymbol{f}_{1,j}) \\
&\quad + \sum_j (\mathtt{sk}^p(\boldsymbol{f}_{1,j}) - \mathtt{sk}(\boldsymbol{f}_{1,j})) \cdot \sum_j (\mathtt{sk}^p(\boldsymbol{f}_{2,j}) - \mathtt{sk}(\boldsymbol{f}_{2,j})).
\end{aligned}$$

By sketch linearity, the first term in the above sum is the estimate of $\boldsymbol{f}_1 \cdot \boldsymbol{f}_2$, which can be bounded by Theorem 2.1 (assuming all sketch computations produce results within their error bounds). Also, by the Cauchy-Schwarz and triangle inequalities, we know that, for any vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$, $|\boldsymbol{v}_i \cdot \boldsymbol{v}_j| \leq \|\boldsymbol{v}_i\| \|\boldsymbol{v}_j\|$ and $\|\sum_i \boldsymbol{v}_i\| \leq \sum_i \|\boldsymbol{v}_i\|$. Combining all these facts, we have:

$$\begin{aligned}
\hat{s}_1 \cdot \hat{s}_2 \in\ & (\boldsymbol{f}_1 \cdot \boldsymbol{f}_2) \pm \epsilon \|\boldsymbol{f}_1\| \|\boldsymbol{f}_2\| \\
& \pm \|\mathtt{sk}(\boldsymbol{f}_2)\| \left\| \sum_j \|\mathtt{sk}^p(\boldsymbol{f}_{1,j}) - \mathtt{sk}(\boldsymbol{f}_{1,j})\right\| \\
& \pm \|\mathtt{sk}(\boldsymbol{f}_1)\| \left\| \sum_j \|\mathtt{sk}^p(\boldsymbol{f}_{2,j}) - \mathtt{sk}(\boldsymbol{f}_{2,j})\right\| \\
& \pm \left(\sum_j \|\mathtt{sk}^p(\boldsymbol{f}_{1,j}) - \mathtt{sk}(\boldsymbol{f}_{1,j})\|\right)\left(\sum_j \|\mathtt{sk}^p(\boldsymbol{f}_{2,j}) - \mathtt{sk}(\boldsymbol{f}_{2,j})\|\right).
\end{aligned}$$

Now, using the special case of Theorem 2.1 for the L_2 norm, and the local site constraint (*), we get:

$$\begin{aligned}
\hat{s}_1 \cdot \hat{s}_2 \in\ & (\boldsymbol{f}_1 \cdot \boldsymbol{f}_2) \pm \epsilon \|\boldsymbol{f}_1\| \|\boldsymbol{f}_2\| \\
& \pm (1+\epsilon) \|\boldsymbol{f}_2\| \frac{\theta}{\sqrt{k_1}} \sum_j \|\mathtt{sk}(\boldsymbol{f}_{1,j})\| \\
& \pm (1+\epsilon) \|\boldsymbol{f}_1\| \frac{\theta}{\sqrt{k_2}} \sum_j \|\mathtt{sk}(\boldsymbol{f}_{2,j})\| \\
& \pm \frac{\theta^2}{\sqrt{k_1 k_2}} \left(\sum_j \|\mathtt{sk}(\boldsymbol{f}_{1,j})\|\right)\left(\sum_j \|\mathtt{sk}(\boldsymbol{f}_{2,j})\|\right) \\
\in\ & (\boldsymbol{f}_1 \cdot \boldsymbol{f}_2) \pm \epsilon \|\boldsymbol{f}_1\| \|\boldsymbol{f}_2\|
\end{aligned}$$

$$\pm(1+\epsilon)^2\|\boldsymbol{f}_2\|\frac{\theta}{\sqrt{k_1}}\sum_j\|\boldsymbol{f}_{1,j}\|$$
$$\pm(1+\epsilon)^2\|\boldsymbol{f}_1\|\frac{\theta}{\sqrt{k_2}}\sum_j\|\boldsymbol{f}_{2,j}\|$$
$$\pm(1+\epsilon)^2\frac{\theta^2}{\sqrt{k_1k_2}}\left(\sum_j\|\boldsymbol{f}_{1,j}\|\right)\left(\sum_j\|\boldsymbol{f}_{2,j}\|\right).$$

Assuming the components of vectors $\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k$ are nonnegative, an application of the Cauchy-Schwartz inequality gives $\sum_{i=1}^k\|\boldsymbol{v}_i\|\le\sqrt{k}\sqrt{\sum_{i=1}^k\|\boldsymbol{v}_i\|^2}\le\sqrt{k}\,\|\sum_{i=1}^k\boldsymbol{v}_i\|$; combining with the above expression, we have:

$$\begin{aligned}\hat{s}_1\cdot\hat{s}_2 &\in \boldsymbol{f}_1\cdot\boldsymbol{f}_2\pm\epsilon\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|\pm2(1+\epsilon)^2\theta\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|\pm(1+\epsilon)^2\theta^2\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|\\ &\in \boldsymbol{f}_1\cdot\boldsymbol{f}_2\pm(\epsilon+(1+\epsilon)^2((1+\theta)^2-1))\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|\end{aligned}$$

In total, we rely on the outcome of $2k_1+2k_2$ sketch computations. Applying a union bound [Motwani and Raghavan 1995], the probability that any of these fails is no more than $2(k_1+k_2)\delta$, giving the required probability bound. □

Thus, by Theorem 3.1, 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 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 $\epsilon,\theta\ll1$). The following corollary gives the adaptation of our tracking result for the special case of a *self-join* query $Q(\boldsymbol{f}_1)=\|\boldsymbol{f}_1\|^2=\sum_v(\boldsymbol{f}_1[v])^2$ (the proof follows from Theorem 3.1 with $\boldsymbol{f}_1=\boldsymbol{f}_2$).

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* $1-2k_1\delta$*,* $\|\hat{s}_1\|^2\in[1\pm(\epsilon+(1+\epsilon)^2\ ((1+\theta)^2-1))]\|\boldsymbol{f}_1\|^2\approx(1\pm(\epsilon+2\theta))\|\boldsymbol{f}_1\|^2$.

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 Dobra et al. [2002]). 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* $\Pi_{i=1}^m \boldsymbol{f}_i \pm (\epsilon\,(2^{m-1}-1)^2 + (1+\epsilon)^m\,((1+\theta)^m - 1))\cdot\, \Pi_{i=1}^m \|\boldsymbol{f}_i\| \approx \Pi_{i=1}^m \boldsymbol{f}_i \pm (\epsilon\,(2^{m-1}-1)^2 + m\theta)\,\Pi_{i=1}^m \|\boldsymbol{f}_i\|$.

PROOF. Proceeding along similar lines as in Theorem 3.1, and using the generalization of Theorem 2.1 to multi-joins (see Section 2.3) as well as an easy generalization of the Cauchy-Schwarz inequality to tensor products of more than two vectors, we have:

$$\begin{aligned}
\Pi_{i=1}^m \hat{s}_i &= \Pi_{i=1}^m \left(\sum_j ((\mathtt{sk}^p(\boldsymbol{f}_{i,j}) - \mathtt{sk}(\boldsymbol{f}_{i,j})) + \mathtt{sk}(\boldsymbol{f}_{i,j}) \right) \\
&\in \Pi_{i=1}^m \boldsymbol{f}_i \pm \epsilon(2^{m-1}-1)^2 \Pi_{i=1}^m \|\boldsymbol{f}_i\| \\
&\quad \pm m\theta(1+\epsilon)^m \Pi_{i=1}^m \|\boldsymbol{f}_i\| \\
&\quad \pm \binom{m}{2} \theta^2 (1+\epsilon)^m \Pi_{i=1}^m \|\boldsymbol{f}_i\| \\
&\quad \cdots \\
&\quad \pm \theta^m (1+\epsilon)^m \Pi_{i=1}^m \|\boldsymbol{f}_i\| \\
&\in \Pi_{i=1}^m \boldsymbol{f}_i \pm \epsilon(2^{m-1}-1)^2 \Pi_{i=1}^m \|\boldsymbol{f}_i\| \pm (1+\epsilon)^m((1+\theta)^m - 1)\Pi_{i=1}^m \|\boldsymbol{f}_i\|.
\end{aligned}$$

The result follows easily from the previous expression and a simple application of the union bound [Motwani and Raghavan 1995]. □

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.[4] 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). Again, we assume a consistent notion of "global time" in our system, and that the dif-

[4]A similar notion of prediction models was introduced for the specific problem of tracking one-dimensional quantiles in Cormode et al. [2005]; instead, we focus on tracking general-purpose randomized sketch summaries of data distributions. Such notions of models are very different from those in Deshpande et al. [2004]: there, models are used in a sensor network to optimize the cost of evaluating one-shot queries by polling specific sensors.

ferences between local "clocks" at sites are sufficiently small to be ignored. 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.

3.2.1 *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,

$$\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) \;=\; \mathtt{sk}^p\big(\boldsymbol{f}^p_{i,j}(t)\big) \;=\; \mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})).$$

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 that are sent when dermined by condition (*)).

3.2.2 *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 $\boldsymbol{f}^p_{i,j}(t) = \frac{t}{t_{prev}}\boldsymbol{f}_{i,j}(t_{prev})$, that is, using a linear scaling of $\boldsymbol{f}_{i,j}(t_{prev})$ to predict the current state 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

$$\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) \;=\; \mathtt{sk}\big(\boldsymbol{f}^p_{i,j}(t)\big) \;=\; \frac{t}{t_{prev}}\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})),$$

a linear scaling of the most recent local sketch of $\boldsymbol{f}_{i,j}$ shipped to the coordinator (and no additional information need be exchanged between sites and the coordinator).

3.2.3 *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 $\Delta = t - t_{prev}$, our velocity/acceleration model predicts the current

state of the $\boldsymbol{f}_{i,j}$ distribution as

$$\boldsymbol{f}^{p}_{i,j}(t) \;=\; \boldsymbol{f}_{i,j}(t_{prev}) + \Delta\boldsymbol{v}_{i,j} + \Delta^2\boldsymbol{a}_{i,j},$$

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 is

$$\mathtt{sk}^{p}(\boldsymbol{f}_{i,j}(t)) \;=\; \mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) + \Delta\mathtt{sk}(\boldsymbol{v}_{i,j}) + \Delta^2\mathtt{sk}(\boldsymbol{a}_{i,j}).$$

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.

Many variations of this scheme are possible. 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). Likewise, it is possible to explicitly send an acceleration sketch instead of computing it implicitly; or to implicitly compute both acceleration and velocity sketches from keeping a history of sketches common to both the coordinator and site j. In our initial experimental evaluation, we found that both these variants performed less well in terms of total communication than the instantiation outlined above, so we focus on this version from now on.

Table II 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).

3.2.4 *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.

Table II. Parameters of Different Prediction Schemes

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	$\mathtt{sk}(\boldsymbol{v}_{i,j})$	$\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) + \Delta\mathtt{sk}(\boldsymbol{v}_{i,j}) + \Delta^2\mathtt{sk}(\boldsymbol{a}_{i,j})$

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, $\epsilon = \frac{\psi}{2}$, $\theta = \frac{\psi}{4}$), the worst-case communication cost for a remote site j processing N_j local updates to stream $\boldsymbol{f}_{i,j}$ is $O(\frac{k_i}{\psi^4}\log(\frac{k_i}{\delta})\log N_j)$.*

PROOF. Firstly, we assume that all updates are insertions, i.e. of the form $\langle i, v, +1\rangle$. In the static model, the worst case effect of each such update is to increase the difference between the predicted (static) distribution and the true distribution by at most 1. Hence, after N_j updates, the L_2 norm is at most N_j. A communication is triggered whenever the norm of the difference is at least a $\frac{\theta}{\sqrt{k_i}}$ fraction of the previous norm of the distribution. The ratio of the squared norm of the new distribution to the old is therefore at least $(1+\frac{\theta^2}{k_i})$, by expanding

$$\begin{aligned}\|\mathtt{sk}(\boldsymbol{f}_{i,j}(t))\|^2 &= \|\mathtt{sk}(\boldsymbol{f}_{i,j}(t)) - \mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) + \mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))\|^2 \\ &> \left(\frac{\theta}{\sqrt{k_i}}\|\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))\|\right)^2 + \|\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t))\|^2.\end{aligned}$$

Thus, the total number of communications is at most $\log_{1+\frac{\theta^2}{k_i}} N_j = O(\frac{k_i}{\theta^2}\ln N_j)$. The cost of each communication is $O(\frac{1}{\epsilon^2}\log 1/\delta)$, proportional to the size of a sketch. So the overall cost depends on $O(\frac{k_i}{\epsilon^2\theta^2})$. To give an error guarantee of $\psi\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|$, we must set $\psi = g_Q(\epsilon, \theta) = \epsilon + (1+\epsilon)^2((1+\theta)^2 - 1) \approx \epsilon + 2\theta$ (here we assume that terms quadratic in ϵ and θ are small enough to be neglected). So we maximize $\epsilon^2\frac{1}{4}(\psi - \epsilon)^2$. Differentiating with respect to ϵ and setting equal to zero gives $\epsilon(\psi - \epsilon)(\psi - 2\epsilon) = 0$. The only feasible solution is $\epsilon = \frac{1}{2}\psi$, and $\theta = \frac{1}{4}\psi$. □

Thus, 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 that each counter in the sketches for site j is of size $O(\log N_j)$ bits.) 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). This requirement, however, could pose significant practical problems when dealing with massive, rapid-rate data streams. Since sketch-summary sizes can vary from tens to hundreds of Kilobytes, especially when tight error guarantees are required, for example, for join or multi-join aggregates [Alon et al. 1999; Dobra et al. 2002], 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 further 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. That is, we improve the update time from $O(\frac{1}{\epsilon^2}\log(1/\delta))$ to $O(\log(1/\delta))$. Our discussion is brief since the structure bears similarities to existing techniques proposed in the context of different (centralized) streaming problems [Charikar et al. 2002; Ganguly et al. 2004], although its application over the basic 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 addition of u to element v, we use the $h_l()$ hash functions to determine the appropriate buckets in the sketch, setting

$$\mathtt{sk}(\boldsymbol{f})[h_l(v), l] \;=\; \mathtt{sk}(\boldsymbol{f})[h_l(v), l] + u\xi_l(v),$$

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. The structure of the sketch is illustrated in Figure 4.

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":

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

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

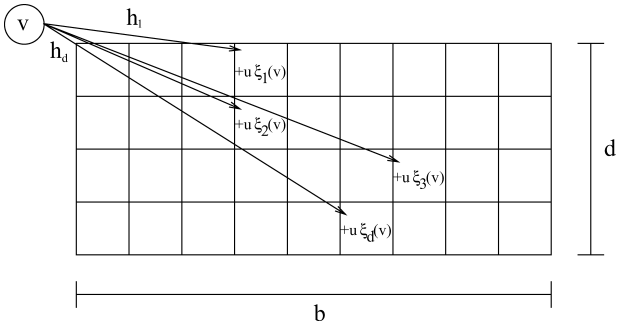


Fig. 4. Structure of the Fast-AGMS Sketch Summary.

time per update (since an element only touches a single partition, i.e., bucket, per hash table).

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* $b = \frac{8}{\epsilon^2}$ *and* $d = 4\log(1/\delta)$*, 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.*

PROOF SKETCH. Consider the estimate X_l given from computing the inner product of the lth row of $\mathtt{sk}(\boldsymbol{f}_1)$ with the corresponding row of $\mathtt{sk}(\boldsymbol{f}_2)$. It can be shown that

$$\mathsf{E}(X_l) = \boldsymbol{f}_1 \cdot \boldsymbol{f}_2 \quad \text{and} \quad \mathsf{Var}(X_l) \leq \tfrac{1}{b}\|\boldsymbol{f}_1\|^2\|\boldsymbol{f}_2\|^2,$$

provided that g_l is drawn from a family of 4-wise independent hash functions, and f_l is drawn from a family of 2-wise independent hash functions. Applying the Chebyshev inequality, $\mathsf{Pr}[|X_l - \boldsymbol{f}_1 \cdot \boldsymbol{f}_2| > \sqrt{\frac{8}{b}}\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|] < \frac{1}{8}$. Taking the median of d such estimators gives an estimate whose probability of being outside this range of $2^{-d/4}$, using standard Chernoff-bound arguments [Motwani and Raghavan 1995]. Thus,

$$\mathsf{Pr}[|\mathtt{sk}(\boldsymbol{f}_1) \cdot \mathtt{sk}(\boldsymbol{f}_2) - \boldsymbol{f}_1 \cdot \boldsymbol{f}_2| > \sqrt{\frac{8}{b}}\|\boldsymbol{f}_1\|\|\boldsymbol{f}_2\|] < 2^{-d/4}.$$

Substituting the values for the b and d parameters gives the required bounds. □

Note that the update cost of our Fast-AGMS sketch remains $O(\log(1/\delta))$ even when tight, *relative-error* guarantees are required for join or multi-join aggregates; in other words, 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). This is crucial since providing tight error guarantees for such complex aggregates can easily imply fairly large sketch summaries [Dobra et al. 2002]. Finally, it is interesting to note that, 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. This difference in space requirements becomes much more pronounced as the ϵ approximation-error bounds become tighter.

Time-Efficient Sketch Tracking. 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 overhead 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.*

PROOF. Our goal is to track $\|\mathtt{sk}^p(\boldsymbol{f}_{i,j}(t)) - \mathtt{sk}(\boldsymbol{f}_{i,j}(t))\|$. We set $\Delta = t - t_{prev}$ and write this quantity out by substituting in the parameters of the acceleration model:

$$\|\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) + \Delta\mathtt{sk}(\boldsymbol{v}_{i,j}) + \frac{1}{2}\Delta^2\mathtt{sk}(\boldsymbol{a}_{i,j}) - \mathtt{sk}(\boldsymbol{f}_{i,j})\|$$

Recall that the estimate of the norm using sketches is produced by computing an estimate from each row of the array of counts, then taking the median of these estimates. We focus on computing the estimate from the lth single row. For this row, we will write V_l for the vector representing the corresponding row from the velocity sketch; A_l for the vector representing the corresponding row from the acceleration sketch scaled by $\frac{1}{2}$; and S_l for the difference of rows coming from $\mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev})) - \mathtt{sk}(\boldsymbol{f}_{i,j})$. The estimate *est* can then be written as

$$\begin{aligned}
est^2 &= \sum_k (S_l[k] + \Delta V_l[k] + \Delta^2 A_l[k])^2 \\
&= \sum_k \big(S_l[k]^2 + 2\Delta S_l[k]V_l[k] + 2\Delta^2(A_l[k]S_l[k] + V_l[k]^2\big). \\
&\quad + \Delta^3 2V_l[k]A_l[k] + \Delta^4 A_l[k]^2\big) \\
&= \sum_k S_l[k]^2 + 2\Delta\sum_k S_l[k]V_l[k] + 2\Delta^2\sum_k A_l[k]S_l[k] \\
&\quad + \Delta^2\sum_k (V_l[k] + \Delta A_l[k])^2
\end{aligned}$$

The last term is easy to track: since V_l and A_l do not change until the sketch tracking condition is violated, we can compute the quantities

$$vv_l = \sum_k V_l[k]^2; \quad aa_l = \sum_k A_l[k]^2; \quad va_l = \sum_k A_l[k]V_l[k]$$

when the sketches are set. At each timestep we will compute this term in constant time:

$$\Delta^2 \sum_k (V_l[k] + \Delta A_l[k])^2 = \Delta^2(aa_l + 2\Delta va_l + \Delta^2 vv_l)$$

The other three terms are affected by updates, but we can use properties of the Fast-AGMS sketches to maintain them efficiently based on their previous values. Define

$$ss_l = \sum_k S_l[k]^2; \quad sv_l = \sum_k S_l[k]V_l[k]; \quad sa_l = \sum_k A_l[k]S_l[k].$$

The structure of Fast-AGMS sketches means that following an update $u \in \{+1, -1\}$ to $\boldsymbol{f}_{i,j}[v]$ only one entry in S_l is affected:

$$S_l[h_l(v)] \leftarrow S_l[h_l(v)] - \xi_l(v) * u.$$

So ss, sv and sa can all be efficiently maintained in constant time per update:

$$\begin{aligned} ss_l &\leftarrow ss_l + (S_l[h_l(v)] - \xi_l(v) * u)^2 - S_l[h_l(v)]^2 \\ &= ss_l + u^2 - 2u * \xi_l(v) * S_l[h_l(v)] \\ sv_l &\leftarrow sv_l - \xi_l(v) * u * V_l[h_l(v)] \\ sa_l &\leftarrow sa_l - \xi_l(v) * u * A_l[h_l(v)] \\ \text{and } S_l[h_l(v)] &\leftarrow S_l[h_l(v)] - \xi_l(v) * u. \end{aligned}$$

Putting all these together, we can rewrite the estimate as

$$est_l^2 = ss_l + 2\Delta sv_l + 2\Delta^2 sa_l + \Delta^2(aa_l + 2\Delta va_l + \Delta^2 vv_l).$$

This allows us to compute the estimate produced by each row in constant time for every update. The estimate for $\|\mathtt{sk}^p(\boldsymbol{f}_{i,j}) - \mathtt{sk}(\boldsymbol{f}_{i,j})\|$ is found by computing the median of all d estimates in time $O(d)$. The total time cost is $O(d)$ per update. □

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.

We need to compare the computed quantity est^2 to $\frac{\theta}{\sqrt{k_i}}\|\mathtt{sk}(\boldsymbol{f}_{i,j})\|$. This can be tracked using the same method that ss is tracked in the above proof. Since S, A and V are stored in $\mathtt{sk}(\boldsymbol{f}_{i,j})$, $\mathtt{sk}(\boldsymbol{v}_{i,j})$ and $\mathtt{sk}(\boldsymbol{a}_{i,j})$, this tracking method requires only constant extra space over the naive scheme that computes the difference every update. To initialize this method, we must compute initial values of vv, va and aa when a new velocity and acceleration sketch is chosen. At this time $t = t_{prev}$ and so $\mathtt{sk}(\boldsymbol{f}_{i,j}(t)) = \mathtt{sk}(\boldsymbol{f}_{i,j}(t_{prev}))$. Hence we initialize $ss = sa = sv = 0$, and proceed to increase these quantities as updates are received. Pseudocode for the tracking procedure is presented in Figure 5.

Procedure SiteUpdate($j, i, v, u = \pm 1, \epsilon, \delta, \theta, k_i, t$)
Input: Site index j, stream index i, inserted/deleted value $v \in [U]$;
sketch error, confidence, and local-deviation parameters ϵ, δ, θ;
"distribution factor" k_i for stream i; current time t .

1. $\Delta := t - t_{prev}$;
2. **for** $l = 1$ **to** d **do**
3. $ss_l := ss_l + u^2 - 2ug_l(v)S_l[h_l(v)]$;
4. $sv_l := sv_l - ug_l(v)V_l[h_l(v)]$;
5. $sa_l := sa_l - ug_l(v)A_l[h_l(v)]$;
6. $est_l^2 := ss_l + 2\Delta sv_l + 2\Delta^2 sa_l + \Delta^2(aa_l + 2\Delta va_l + \Delta^2 vv_l)$;
7. $skx_l^2 := skx_l^2 + 2ug_l(v)S_l[h_l(v)] + u^2$
8. **if** ($\text{median}_l\{est_l^2\} > \frac{\theta^2}{k_i}\,\text{median}_l\{skx_l^2\}$) **then**
9. compute new $\mathtt{sk}(\boldsymbol{v}_{i,j})$, $\mathtt{sk}(\boldsymbol{a}_{i,j})$;
10. send new model to coordinator;
11. **for** $l = 1$ **to** d **do**
12. recompute aa_l, va_l, vv_l;
13. $t_{prev} = t$;

Fig. 5. Fast procedure for tracking updates at remote sites.

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 [Gilbert et al. 2001]), 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 $a \le v \le b$, 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 [Charikar et al. 2002]).

Histogram and Wavelet Representations. Gilbert et al. [2001] 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 a $g(\epsilon, \theta)$-approximate sketch; thus, our analysis in conjunction with Theorem 3 of Gilbert et al. [2001], 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. [2002] 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.[5] Combining our results with Theorem 3 of [Thaper et al. 2002], we have a scheme for tracking a $1 + Bg(\epsilon, \theta)$ approximation to the best B-bucket multi-dimensional histogram.

4. EXTENSIONS

We have so far considered the case where queries are to be answered on the whole history of updates. In many applications, only recent updates are relevant to queries, and older information should be dropped or weighted so that its contribution is minimal. We consider two standard approaches to keeping results fresh, and show how to fit them into our tracking scenario. We also discuss the extension of our techniques to more complex, *multilevel* distributed monitoring hierarchies, and analyze the optimal approximation parameter settings under different communication-cost objectives. Lastly, we consider alternate sketch prediction models.

4.1 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 $\langle i, v, -1\rangle$ 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.

In the case that the remote site does not have space to buffer all updates in the sliding window, techniques such as the exponential histogram approach [Datar et al. 2002] can be applied to bound the amount of space required. This method allows the sketch of the window to be approximated by the combination of a logarithmic number of sketches of nonoverlapping subwindows. However, this does not directly lead to guaranteed bounds: although the sketch of the sliding window is well approximated by this method, when the predicted sketch is subtracted, the bounds do not hold. In practice, this approach or similar methods of dividing the window into nonoverlapping windows and approximating the window with these pieces is likely to give sufficiently good results. An alternate sliding window approach is to consider only the most recent SW updates. This has two variants: when this policy is applied locally at each site, and when

[5]Although this procedure will typically return more than B buckets [Thaper et al. 2002].

the policy is to be applied globally over all updates. In the first case, the above discussion applies again, and the same results follow. For the second case, the problem seems more involved, and can provide an interesting direction for future research.

The exponential decay model is a popular alternative to the sliding window model [Gilbert et al. 2001]. Briefly, the current distribution $\boldsymbol{f}_i(t)$ is computed as $\boldsymbol{f}_i(t) = \lambda^{t-t_{prev}} \boldsymbol{f}_i(t_{prev})$ 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: $\mathtt{sk}(\boldsymbol{f}_i(t)) = \lambda^{t-t_{prev}} \mathtt{sk}(\boldsymbol{f}_i(t_{prev}))$ (where t_{prev} denotes the time of the last update). 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.

We outline the method briefly. For each entry of the array of counts in $\mathtt{sk}(\boldsymbol{f}_{1,j})$, we additionally keep a tag denoting the time t when it was last updated. To apply the decay to the sketch for a time $t - t_{prev}$, we take the estimate at time t_{prev} and multiply it by $\lambda^{t-t_{prev}}$. To process an update u to location i at time t, in each row we identify the entry in S that is affected ($S_l[h_l(v)]$), and look up the last time this entry was probed as t'. We can then update our estimate by setting

$$est_l^2 \leftarrow est_l^2 + u^2 + 2\lambda^{t-t'} * u * \xi(v)S[h_l(v)]$$

(this expression comes from computing the change due to decaying the entry of S by $\lambda^{t-t'}$, and subtracting the contribution of this; and then adding on the contribution from the addition of u). We update

$$S_l[h_l(v)] \leftarrow \lambda^{t-t'} S_l[h_l(v)] + u * \xi_l(v),$$

and set the time of last modification to t. From this, we have

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.*

4.2 Approximate Hierarchical Query Tracking

Consider a more complex distributed-tracking scenario where the communication network is arranged as a *tree-structured* hierarchy of nodes (i.e., sites)—our goal here is for the root node to effectively track an approximate query answer over the update streams observed at the leaf nodes of the hierarchy. (To simplify the discussion, we assume that internal nodes in the hierarchy do not observe any updates; however, such generalizations can be incorporated into our model by adding dummy leaf-child nodes to internal nodes to accept their corresponding streams.) This hierarchical-monitoring architecture generalizes the flat, single-level model discussed earlier in this paper (essentially, a

one-level hierarchy with remote sites as leaves). Such monitoring hierarchies arise naturally, for instance, in the context of sensor networks, where sensor nodes are typically organized in a routing tree with a root base-station monitoring the sensornet operation [Madden et al. 2003]. In such scenarios, naively propagating sketch updates from the leaves to the root node is wasteful; distribution changes at leaf nodes of a subtree may effectively "cancel out" at a higher node in the hierarchy rendering further communication with nodes higher in the hierarchy unnecessary. For such multilevel hierarchies, our tracking scheme should be able to exploit stability properties *at any node of the hierarchy*.

In this section, we demonstrate how our earlier ideas and results can be used to effectively solve and analyze approximate query-tracking problems in the case of such general hierarchies. For simplicity, we concentrate on the case of a *self-join* query $Q(\boldsymbol{f}) = \|\boldsymbol{f}\|^2$ (Corollary 3.2), where the update stream $\boldsymbol{f}$ is observed across all leaf nodes in the hierarchy; the extensions to handle more general queries and site subsets are straightforward.

Assume that our tracking hierarchy comprises $h + 1$ levels, with the root node at level 0 and the leaf nodes at level h. We can compute an approximate sketch over the union of streams observed at the leaf nodes by running our sketch-tracking scheme between each internal node and its children. That is, each internal node u tracks an (approximate) AGMS sketch over its children, and then passes up relevant information to its parent when its locally-observed sketch (which summarizes the data distribution of all streams in its subtree) violates a specified deviation bound with respect to its corresponding prediction at u's parent (i.e., condition (*) with k_i equal to the number of siblings of u).

Just as in the flat, single-level case it suffices to allocate the same deviation tolerance θ to every remote site, we argue that it suffices to allocate the same θ_l parameter to every node at level l in the hierarchy—essentially, this implies that each level-$(l-1)$ node gives all its children the maximum possible error tolerance (based on its own error bounds) in order to minimize communication cost across levels $l-1$ and l. Now consider a node u at level l in the tree hierarchy and let S_u denote the union of update streams in the subtree rooted at u, and define (1) α_l as the accuracy at which u tracks its local sketch summary (for the S_u stream); and, (2) θ_l as the bound on the deviation of locally maintained sketches (with respect to their predictions) at node u. The following corollary then follows easily from Corollary 3.2.

COROLLARY 4.2. *Let α_l and θ_l be as defined above. Then, the compounded error of the local AGMS sketch summaries for nodes at level $l-1$ of the hierarchy is $\alpha_{l-1} = \alpha_l + 2\theta_l$.*

Corollary 4.2 essentially allows us to "cascade" our basic, single-level tracking scheme to the case of multilevel hierarchies. Specifically, assuming a fixed AGMS sketching error ϵ at the leaf nodes of the hierarchy, then, by Corollary 4.2, summing across all levels, the total sketch-tracking error at the root node is $\alpha_0 = \epsilon + 2\sum_{l=1}^{h}\theta_l$.

Assuming that the sketching error ϵ at leaf nodes is fixed (e.g., based on site memory limitations), we now seek to optimize the settings for the θ_l parameters

for minimizing communication costs. We consider the worst-case bounds for our static-prediction model, and two possible optimization objectives: (1) the *maximum transmission cost* for any node in the hierarchy (or, equivalently, the maximum load on any communication link), and (2) the *aggregate communication cost* over the entire communication hierarchy. Both of the above objectives are important in the sensornet context (e.g., for maximizing the lifetime of a sensor network) as well as more traditional distributed network-monitoring scenarios. To simplify the analysis that follows, we assume a *regular* hierarchical-monitoring topology, where both (a) the branching factor (i.e., number of siblings), and (b) the number of observed updates for a node at level l are fixed at k_l and N_l, respectively. (Our analysis can also provide some guidance for effective heuristics for setting the deviation parameters in more general scenarios.) From the analysis in Section 3.2.4, the (worst-case) transmission cost for a node at level l is $O(\frac{\sqrt{k_l}}{\theta_l^3} \log(\frac{k_l}{\delta}) \log N_l)$.

Maximum Transmission Cost Minimization Problem. Determine θ_l's that minimize $\max_l\{\frac{\sqrt{k_l}}{\theta_l^3} \log(\frac{k_l}{\delta}) \log N_l\}$ subject to the total-error constraint $\sum_l \theta_l = \theta$.

For this minimization problem, it is not difficult to see that the optimal point occurs when the per-node transmission costs at all levels are equal, giving the optimal per-level θ_l settings

$$\theta_l = \frac{\theta \left(\sqrt{k_l} \log\left(\frac{k_l}{\delta}\right) \log N_l\right)^{1/3}}{\sum_j \left(\sqrt{k_j} \log\left(\frac{k_j}{\delta}\right) \log N_j\right)^{1/3}}.$$

In the case of minimizing total communication, the per-node transmission cost at level l is multiplied by the total number of nodes at that level $K_l = \Pi_{j=1}^{l} k_j$, and we have a sum objective function. This is a more complicated minimization problem, but we show a closed-form solution for the optimal θ_l settings.

Aggregate Communication Cost Minimization Problem. Determine θ_l's that minimize the sum $\sum_l \frac{K_l \sqrt{k_l}}{\theta_l^3} \log(\frac{k_l}{\delta}) \log N_l$ subject to the total-error constraint $\sum_l \theta_l = \theta$.

THEOREM 4.3. *The optimal θ_l values for minimizing the (worst-case) aggregate communication cost over a (regular) multi-level tracking hierarchy are given by*

$$\theta_l = \frac{\theta \left(K_l \sqrt{k_l} \log\left(\frac{k_l}{\delta}\right) \log N_l\right)^{1/4}}{\sum_j \left(K_j \sqrt{k_j} \log\left(\frac{k_j}{\delta}\right) \log N_j\right)^{1/3}}.$$

PROOF. Let $c_l = K_l \sqrt{k_l} \log(\frac{k_l}{\delta}) \log N_l$, for all levels l. Our proof uses *Hölder's inequality* [Hardy et al. 1988], which states that, for any $x_l, y_l \geq 0$, and $p, q > 1$

such that $\frac{1}{p} + \frac{1}{q} = 1$:

$$\left(\sum_l x_l^p\right)^{1/p} \cdot \left(\sum_l y_l^q\right)^{1/q} \geq \sum_l x_l\, y_l,$$

with equality holding only if $y_l = \lambda \cdot x_l^{p-1}$ for all l.

Substituting $p = 4, q = 4/3, x_l = (\frac{c_l}{\theta_l^3})^{1/4}$, and $y_l = \theta_l^{3/4}$ in Hölder's Inequality, we get

$$\left(\sum_l \frac{c_l}{\theta_l^3}\right)^{1/4} \cdot \left(\sum_l \theta_l\right)^{3/4} \geq \sum_l c_l^{1/4},$$

or, equivalently (since $\sum_l \theta_l = \theta$), $\sum_l \frac{c_l}{\theta_l^3} \geq \frac{1}{\theta^3}\left(\sum_l c_l^{1/4}\right)^4$.

Note that the left-hand side of this inequality is precisely our optimization objective, whereas the right-hand side is constant. Thus, the optimal (i.e., minimum) value for our objective occurs when equality holds in this instance of Hölder's inequality, or, equivalently, if $\theta_l^{3/4} = \lambda(\frac{c_l}{\theta_l^3})^{3/4}$, which after some simplification, gives $\theta_l = \lambda' c_l^{1/4}$ (where the new proportionality constant is $\lambda' = \lambda^{1/3}$). Coupled with the total error constraint $\sum_l \theta_l = \theta$, this directly implies that the optimal θ_l values are given by $\theta_l = \theta c_l^{1/4} / \sum_j c_j^{1/4}$. The result follows. □

4.3 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. Our main focus has been on defining the framework which guarantees correctness, and we show in our experimental simulations that even simple models achieve significant savings. The choice of model is orthogonal to the framework, and naturally given more knowledge of the application area, one can select any model which can predict the distribution well. Note though that there is tradeoff here: the more complex the model, the more parameters that must be communicated between the site and the coordinator, which can lead to higher communication cost; thus, better prediction does not necessarily immediately translate into lower overall communication cost.

One can easily envisage other models of evolving local distributions beyond those discussed here, and translating these into predicted sketches by applying the linearity properties of the sketch transformation. In particular, the linear predictions can be seen as simple instances of the more general class of ARMA models [Box and Jenkins 1970]. Direct implementation of these models requires some adjustment, though: these models typically forecast ahead only a small number of timesteps, whereas to ensure limited communication, we need to forecast over many thousands or even millions of timesteps. A

second consideration is that whereas such models typically use least-squares regression to minimize the *error* in prediction, we have a distinct objective: to minimize the *frequency* with which the prediction error exceeds a given threshold. Nevertheless, such models have been successfully used in similar settings such as sensor networks [Tulone and Madden 2006], and so could be applied here.

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 shipping only information on some of the heavy-hitter items, instead of 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 "skimmed sketches" of Ganguly et al. [2004], but for the purpose of decreasing communication rather than increasing accuracy. Exploring the applicability and potential benefits of such sketch-skimming approaches for our distributed query-tracking problem is an interesting direction for future work in this area.

5. EXPERIMENTAL STUDY

We conducted an experimental study on the proposed tracking algorithms, to understand the effect of setting various parameters (ϵ, θ, and window W for the velocity/acceleration model), to evaluate the communication savings from our method, and to measure the accuracy and time cost of our methods compared to the baseline solution of each remote site communicating every update to the coordinator site. We also tested the overall accuracy of our approximate methods by comparing to the exact answer for various queries, and looked at the time benefits of using the fast update techniques we have introduced.

5.1 Testbed and Methodology

We implemented a test system that simulated running our protocols in C.[6] Throughout, we set the probability of failure, $\delta = 1\%$. 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 (static, linear-growth or velocity/acceleration) with the same tracking parameters ϵ, θ. We implemented various natural optimizations of our methods. When each site has to communicate to the coordinator, it computes whether it is more efficient to send a sketch or to send the updates since the last communication, and sends the cheaper message. Since the coordinator has

[6]Our implementation of Fast-AGMS sketches was modified from our implementations available at `http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`.

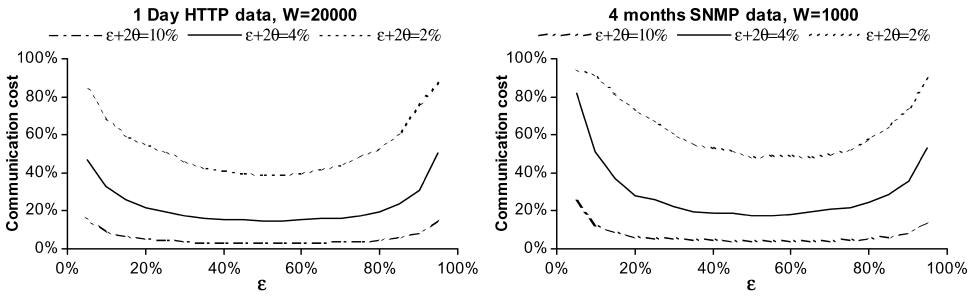
the previous sketch, it can compute the new sketch by adding on the recent updates, so both remote site and coordinator stay in lockstep. This ensures that the cost is never worse than the trivial solution of sending every update to the coordinator, and is important in the early stages in the protocol, when the site is still learning the nature of its streams, and so must update the coordinator often.

We report the results of experiments run on two data sets. The first was drawn from the Internet Traffic Archive (`http://ita.ee.lbl.gov/`), representing HTTP requests sent to servers hosting the World Cup 1998 Web Site. Servers were hosted in four geographic locations: Plano, Texas; Herndon, Virginia; Santa Clara, California; and Paris, France. 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 "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 nonuniform distribution. The second data set consisted of SNMP network usage data obtained from CRAWDAD (the Community Resource for Archiving Wireless Data at Dartmouth, `http://cmc.cs.dartmouth.edu/data/dartmouth.html`). It consists of measurements of total network communication every five minutes over a four month period at a large number of different access points (approximately 200). We divided these access points into eight groups to create a data set with eight sites and 6.4 million total requests. Here, we used the "size" attribute as the one to index the data on, since this takes a very large number of values, and will be challenging to predict the distribution accurately. 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 32-bit 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; again, our goal is to see this error as low as possible.

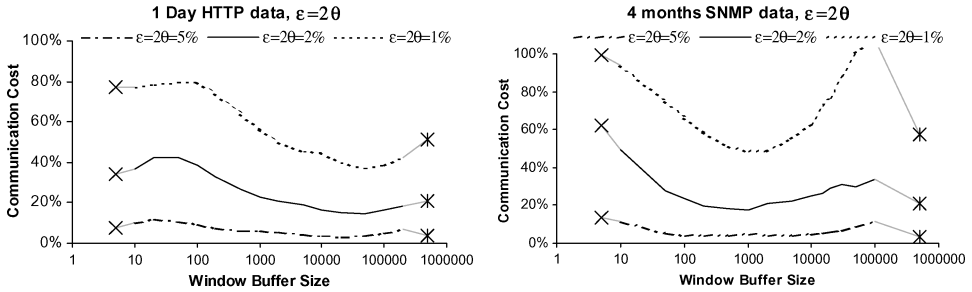
5.2 Experimental Results

Setting Parameters and Tradeoffs. First, 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, and the complete SNMP data. Figure 6 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 0.55–0.6ψ. Our analysis in Section 3.2 showed that for a worst case distribution under the static model, ϵ



(a) θ–ϵ tradeoff for HTTP data. (b) θ–ϵ tradeoff for SNMP data.

Fig. 6. Tradeoff between the parameters ϵ and θ.

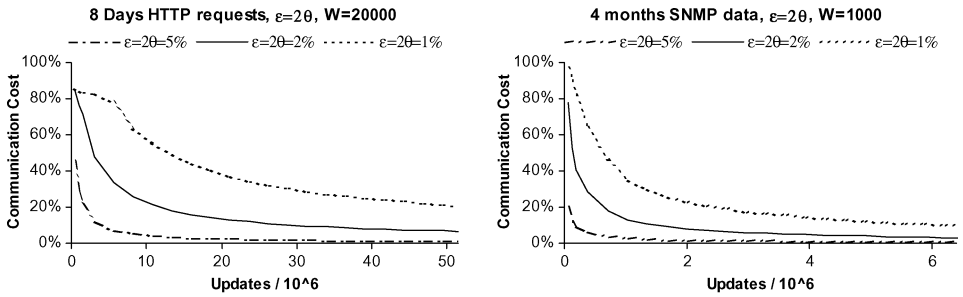


(a) Varying window size on HTTP data. (b) Varying window size on SNMP data.

Fig. 7. Effect of varying the window size used to estimate the "velocity" sketch.

should be around $\psi/2$. 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 $W = 20000$ on the HTTP data and $W = 1000$ on the SNMP data; 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 7, we show the effect of varying the window size W for the velocity/acceleration model on the communication cost for three values of $\psi = \epsilon + 2\theta$ on both data sets. 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, hence velocity and acceleration components are set to zero. 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. On the HTTP World Cup data (Figure 7(a)), 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. For each curve, the least cost is between half and a third of the greatest cost for that curve. The static model gets close to the worst cost, while the linear growth model does quite well, but still about



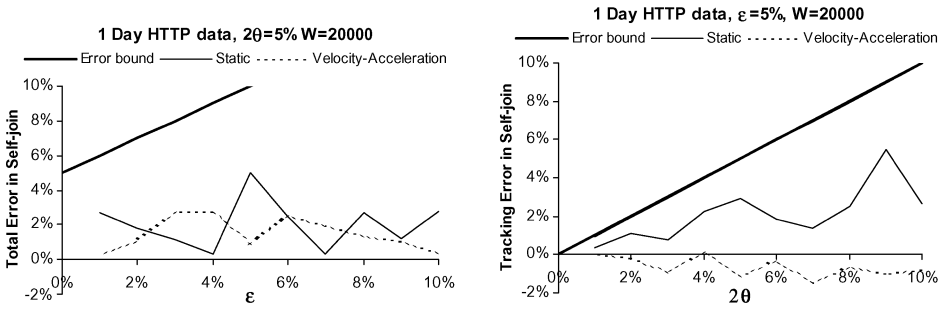
(a) Communication cost for HTTP data. (b) Communication cost for SNMP data.

Fig. 8. Communication cost as the number of updates increases.

a third more than the best velocity/acceleration model. For the HTTP data set, we see that irrespective of the $g(\epsilon, \theta)$ value the best setting of W is in the range 10000–100000. We observe similar behavior on the SNMP data, although the benefits of using a window over the linear growth model decrease as ϵ decreases. For the remainder of our experiments, we focus on the velocity/acceleration model with $W = 20000$ for the HTTP data, and $W = 1000$ for the SNMP data.

Communication Cost. We look at how the communication cost evolves with time in Figure 8, 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 (with $k = 4$ as before), and on the same Dartmouth SNMP data set treated as updates to a single site (so $k = 1$). For both data sets, the behavior is fairly similar. 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, that is, where no communication is necessary. This implies that in the long term our methods reach a "steady state," where no communication is necessary, and large savings result over shipping up every update to the coordinator.

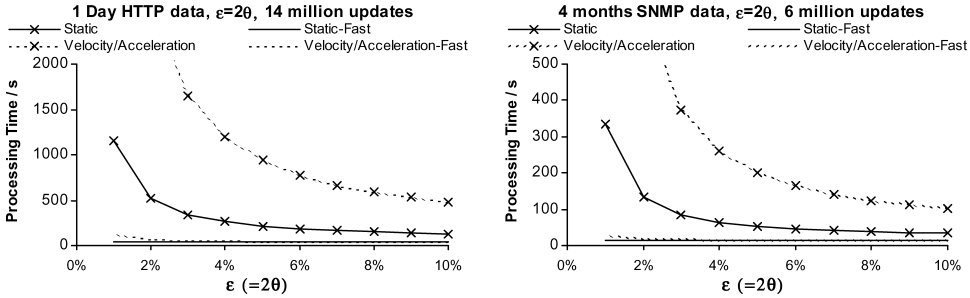
Accuracy of Approximate Query Answers. Our first set of experiments focused on the communication cost of our proposed protocols. We now consider the accuracy they provide for answering queries at the coordinator, and the time cost at the remote sites. In Figure 9(a), we plot the error in answering queries 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 $\epsilon + 2\theta \approx g(\epsilon, \theta)$: all the results fall well within this bound. Both static and velocity/acceleration models give similar errors at the coordinator. Note that there some variability in the error with different values of ϵ, which arises from two sources: (1) variation due to the sketch error bound ϵ. (2) variation from the tracking bound θ. Depending on when the query is posed, the remote site may be using little of the "slack" that this bound gives, or it may be using almost all of it. Therefore, we should



(a) Estimation quality, fixed θ, varying ϵ.

(b) Estimation quality due to tracking delay with sketch error subtracted, as θ varies.

Fig. 9. Experiments evaluating the quality of the returned results.



(a) Timing cost for HTTP data.

(b) Timing cost for SNMP data.

Fig. 10. Timing cost, comparing fast tracking methods to performing sketch estimation every step, for static and acceleration models.

not expect to see any overall trend as ϵ varies, beyond that the total error is within the global guarantee.

In Figure 9(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 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, thus guaranteeing that it meets the error bound.

Timing Results. Lastly, we consider the time cost of our tracking methods. 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 10). The communication cost and accuracy of both versions was

the same in all cases: our fast tracking techniques compute the same functions, but are designed to work more quickly. The time cost is composed of the time required to update each sketch, and the time to recompute the sketch model when the error bounds are exceeded. For the "slow" implementation, the former requires time linear in the size of the sketch, which is in turn quadratic in $1/\epsilon$; for the "fast" version, the cost is independent of ϵ. As can be seen in the diagram, the direct implementations of the tracking methods rise sharply as ϵ approaches zero, while the fast implementations hardly vary. For small ϵ, the fast velocity/acceleration method becomes more expensive since, while update operations are still fast, recomputing the sketches when tracking bounds are broken begins to contribute more significantly to the overall cost. For $\epsilon \geq 3\%$ on the World Cup HTTP data (Figure 10(a)) the cost was 36 seconds in the static case, and 50 seconds for the more complex velocity/acceleration model to process all 14 million updates. This gives an effective processing rate of around 300–400 thousand updates per second per site; equivalently, an average overhead of 3 microseconds per update on our experimental setup (2.4 GHz Pentium desktop). A similar rate was observed on the Dartmouth SNMP data (Figure 10(b)).

Experimental Conclusions. Our experiments show that significant communication savings are possible for the variety of tracking problems based on the key sketch tracking problem that we address. With an approximation factor of 10%, the communication cost can be less than 3% of sending every update directly to the coordinator, and this saving increases as more updates are processed. Time overhead is minimal, a few microseconds to update the necessary tracking structures, and typically a few kilobytes per sketch, plus space to store a recent history of updates. Our more detailed sketch prediction models seem to offer significant improvements over the simplest model. 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 (one could imagine a more involved algorithm that tries several values of W in parallel and eventually settles on the one that minimizes communication). Failing this, linear growth provides adequate results, and requires no extra parameters to be set.

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.

The main results show that join, multi-join and self-join size queries can be answered with guaranteed error bounds provided remote sites track conditions that depend only on individual streams observed locally. With appropriate models predicting future behavior based on a collected history, little communication is needed between the remote sites and the coordinator site. A wide range of queries can be answered by the coordinator: essentially, any query that can be

approximated using ϵ-approximate sketches can now be answered with $g(\epsilon, \theta)$ error, including heavy hitters, wavelets, and multidimensional histograms.

REFERENCES

ALON, N., GIBBONS, P. B., MATIAS, Y., AND SZEGEDY, M. 1999. Tracking join and self-join sizes in limited storage. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Philadeplphia, PA.

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*. Philadelphia, PA, 20–29.

BABCOCK, B. AND OLSTON, C. 2003. Distributed top-K monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA.

BOX, G. AND JENKINS, G. 1970. *Time Series Analaysis: Forecasting and Control*. Holden-Day.

CHARIKAR, M., CHEN, K., AND FARACH-COLTON, M. 2002. Finding frequent items in data streams. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Malaga, Spain.

CHU, D., DESHPANDE, A., HELLERSTEIN, J. M., AND HONG, W. 2006. Approximate data collection in sensor networks using probabilistic models. In *Proceedings of the 22nd International Conference on Data Engineering*. Atlanta, GA.

CORMODE, G. AND GAROFALAKIS, M. 2005. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st International Conference on Very Large Data Bases*. Trondheim, Norway.

CORMODE, G., GAROFALAKIS, M., MUTHUKRISHNAN, S., AND RASTOGI, R. 2005. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Baltimore, MD.

CORMODE, G. AND MUTHUKRISHNAN, S. 2003. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. San Diego, CA, 296–306.

CORMODE, G. AND MUTHUKRISHNAN, S. 2004. An improved data stream summary: The count-min sketch and its applications. *Latin American Informatics*. 29–38.

CRANOR, C., JOHNSON, T., SPATSCHECK, O., AND SHKAPENYUK, V. 2003. Gigascope: A stream database for network applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA.

DAS, A., GANGULY, S., GAROFALAKIS, M., AND RASTOGI, R. 2004. Distributed set-expression cardinality estimation. In *Proceedings of the 30th International Conference on Very Large Data Bases*. Toronto, Canada.

DATAR, M., GIONIS, A., INDYK, P., AND MOTWANI, R. 2002. Maintaining stream statistics over sliding windows. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*. San Francisco, CA, 635–644.

DESHPANDE, A., GUESTRIN, C., MADDEN, S. R., HELLERSTEIN, J. M., AND HONG, W. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the 30th International Conference on Very Large Data Bases*. Toronto, Canada.

DOBRA, A., GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Madison, WI, 61–72.

GANGULY, S., GAROFALAKIS, M., AND RASTOGI, R. 2003. Processing set expressions over continuous update streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA.

GANGULY, S., GAROFALAKIS, M., AND RASTOGI, R. 2004. Processing data-stream join aggregates using skimmed sketches. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT'04)*. Heraklion-Crete, Greece.

GIBBONS, P. B. 2001. Distinct sampling for highly accurate answers to distinct values queries and event reports. In *Proceedings of the 27th International Conference on Very Large Data Bases*. Roma, Italy.

GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. J. 2001. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*. Roma, Italy.

GREENWALD, M. B. AND KHANNA, S. 2001. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Santa Barbara, CA.

GREENWALD, M. B. AND KHANNA, S. 2004. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Paris, France.

HARDY, G., LITTLEWOOD, J., AND PÓLYA, G. 1988. *Inequalities, 2nd Ed.* Cambridge University Press.

HUANG, L., GAROFALAKIS, M., JOSEPH, A. D., AND TAFT, N. 2007a. Communication-efficient tracking of distributed cumulative triggers. In *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS'07)*. Toronto, Canada.

HUANG, L., NGUYEN, X., GAROFALAKIS, M., HELLERSTEIN, J. M., JORDAN, M. I., JOSEPH, A. D., AND TAFT, N. 2007b. Communication-efficient online detection of network-wide anomalies. In *Proceedings of IEEE INFOCOM*. Anchorage, AK.

JAIN, A., CHANG, E. Y., AND WANG, Y.-F. 2004. Adaptive stream resource management using Kalman filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Paris, France.

KERALAPURA, R., CORMODE, G., AND RAMAMIRTHAM, J. 2006. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Chicago, IL, 289–300.

MADDEN, S. R., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2003. The design of an acquisitional query processor for sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA.

MANJHI, A., SHKAPENYUK, V., DHAMDHERE, K., AND OLSTON, C. 2005. Finding (recently) frequent items in distributed data streams. In *Proceedings of the 21st International Conference on Data Engineering*. Tokyo, Japan.

MANKU, G. S. AND MOTWANI, R. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*. Hong Kong, China, 346–357.

MOTWANI, R. AND RAGHAVAN, P. 1995. *Randomized Algorithms*. Cambridge University Press.

OLSTON, C., JIANG, J., AND WIDOM, J. 2003. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA.

SHARFMAN, I., SCHUSTER, A., AND KEREN, D. 2006. A geometric approach to monitoring threshold functions over distributed data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Chicago, IL, 301–312.

THAPER, N., GUHA, S., INDYK, P., AND KOUDAS, N. 2002. Dynamic multidimensional histograms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Madison, WI, 428–439.

TULONE, D. AND MADDEN, S. 2006. Paq: Time series forecasting for approximate query answering in sensor networks. In *the European Conference on Wireless Sensor Networks (EWSN)*.

Received January 2007; revised August 2007; accepted January 2008