What's Different: Distributed, Continuous Monitoring of Duplicate-Resilient Aggregates on Data Streams

Graham Cormode
Bell Laboratories
cormode@bell-labs.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Wei Zhuang
Rutgers University
weiz@paul.rutgers.edu

Abstract

Emerging applications in sensor systems and network-wide IP traffic analysis present many technical challenges. They need distributed monitoring and continuous tracking of events. They have severe resource constraints not only at each site in terms of per-update processing time and archival space for high-speed streams of observations, but also crucially, communication constraints for collaborating on the monitoring task. These elements have been addressed in a series of recent works.

A fundamental issue that arises is that one cannot make the "uniqueness" assumption on observed events which is present in previous works, since widescale monitoring invariably encounters the same events at different points. For example, within the network of an Internet Service Provider packets of the same flow will be observed in different routers; similarly, the same individual will be observed by multiple mobile sensors in monitoring wild animals. Aggregates of interest on such distributed environments must be resilient to duplicate observations.

We study such duplicate-resilient aggregates that measure the extent of the duplication—how many unique observations are there, how many observations are unique—as well as standard holistic aggregates such as quantiles and heavy hitters over the unique items. We present accuracy guaranteed, highly communication-efficient algorithms for these aggregates that work within the time and space constraints of high speed streams. We also present results of a detailed experimental study on both real-life and synthetic data.

1 Introduction

In recent years, a variety of large-scale monitoring applications have arisen. A key example is *network-wide* monitoring of the IP network, where network management requires collecting data from multiple routers and locations, collating them, performing real-time analyses for network operations, quality monitoring, security, providing service-level guarantees, and so on. Another example is *wildlife monitoring* of animals in the wild such as the ZebraNet [19] project where collars containing a variety of sensors are placed on Zebras in the wild and fixed and mobile basestations (some on elephants!) are used to monitor the herds.

Such large scale monitoring applications represent many challenges to data management and require new approaches to cope with them. For example, they are inherently distributed over a wide area, and rely on a possibly unreliable communication network. Data collected is massive and typically is presented as a "stream" of (unsorted) readings. Further, there are severe resource constraints in

• *space*: routers typically do not have high speed memory capable of capturing packet logs for more than a few minutes; sensors on zebras have a few Mb of memory and can not store more than a few minutes of readings from accelerometer, GPS, and other sensors.

• *processing power*: much of router CPU is devoted to packet forwarding and flow management and only a small portion is available for logging or monitoring, a portion that goes down significantly in times of greatest need such as during Denial of Service Attacks when the number of flow increases sharply and more of the CPU time is needed to manage the traffic; sensors have modest CPUs with low power requirements and a significant portion of the processing is devoted to radio management for communication, and

• *communication capability*: communicating entire packets logs will be a significant fraction of network capacity and is rarely done in large Internet Service Providers; radio communication in sensors is a power-hog, can be spotty and sporadic, and is used only sparingly.

In addition, the challenges in large scale monitoring are also conceptual, related to query processing.

Nature of query processing. Rather than *on-demand*, user-driven queries, specified ad hoc by users in SQL for immediate processing as is common in Database Management Systems, the monitoring is typically *continuous*. For example, in an IP network, one task is to continuously observe the traffic patterns and cause an alert (or take some action) when a Denial of Service attack is detected.

While continuous queries present challenges even within a standard DBMS context, for large-scale monitoring, the challenges are deeper. The volume of the data collected means that the simple solution of pushing all data back to a single coordinator node for centralized processing does not scale. Instead, the goal must be to minimize communication to the extent possible.

The emerging approach to supporting continuous queries in distributed monitoring applications is to allow approximation and do *in-network aggregation*. That is, the monitoring query is compiled into a set of constraints that are pushed down into the monitoring nodes. These create concise summaries of the data observed, and send the summaries to coordinator nodes, which collate and recompute constraints, guaranteeing that the monitoring problem is accurately answered at the coordinator, up to some approximation criterion.

Nature of Queries. Sophisticated monitoring queries tend to be holistic, requiring features from multiple attributes at multiple locations to be coordinated and compared. For example, in wildlife monitoring, a natural query is to track the *herd size* comprising members that form a social community that is distinct from merely being at a waterhole at discrete times. In IP routers, a natural query is to track total traffic volume of traffic that is generated inside a network and is destined to the outside. Hence, distributed monitoring queries go beyond queries that can be easily decomposed into aggregates at each monitoring point, and aggregated via select, project, SUM, COUNT etc.

One of the chief characteristics of large scale monitoring systems is that the same "event" may be observed multiple times at multiple locations. In an IP network, the same packet may be seen at many tap points within the network, or by only one, depending on how the packet is routed. In sensor networks, each meeting between two members of the herd may be registered by multiple sensors, but should only be counted once; for example, in tracking wild animal interactions it is important to distinguish between previously observed groupings and new ones, which may happen in different locations or at different times. Further, the communication itself may be lossy, so in-network aggregation must be resilient to information that is repeated to avoid loss of data. For example, TCP retransmits lost packets and leads to the same packet being seen even at a given monitor more than once. In order to account for sporadic sampling and provide reliability, wildlife monitoring systems such as the ZebraNet rely on periodic exchange of data between members so that when data is collected from multiple members, there is significant repetition of same events. Other sensor systems also see such duplications [6, 23]. Hence we must support tracking of events in a way that is *duplicate-resilient* so duplication of messages and readings does not affect the overall accuracy of the monitoring task.

In this paper, we study *duplicate-resilient aggregates* in the continuous, distributed scenario setting for large-scale monitoring. In particular, we propose two problem classes:
1. *What is the amount of duplication in the network?* We study direct aggregates that quantify the amount of duplication in the system, such as tracking the number of distinct events seen, or how many events are unique (i.e. are not duplicated), or more generally, what is the duplication factor for each event.
2. *What are the versions of classical aggregates in the presence of duplicates?* Here, we study the indirect aspects of duplications by studying classical *holistic* aggregates—like quantiles, heavy hitters—and more generally various selectivity queries over distinct events.

More precisely, our contributions are as follows.
- We formalize duplicate-resilient monitoring in a continuous, distributed setting.
- We identify two fundamental primitives—distinct counting and distinct sampling—at the heart of a wide variety of more complex queries that are duplicate-resilient, and present efficient solutions for tracking them in the continuous, distributed setting within resource constraints—per-item processing time and limited space—at each monitored site. Our solutions give trade-offs between the communication cost and the accuracy of the monitored query at a central site. In addition, they show the trade-offs of each site monitoring aggregates with and without global knowledge of the data distribution at other sites, and the trade-offs involved in information flow between the central monitoring site and the monitored sites.
- We show applications of the primitives above to several duplicate-resilient aggregate monitoring problems.
- We perform an extensive experimental evaluation of our methods. This highlights the differences between the design choices across our techniques, and points to which approaches are generally the most successful at minimizing the communication burden of the monitoring task beyond the asymptotic costs found from our analysis.

2 Related Work

There has been a lot of work recently on estimating various aggregates on data "streams" that work within our limited space and per-item processing concerns, motivated by IP traffic, financial, web click streams and others. These methods are surveyed in [2, 22]. There has been tremendous progress in building Data Stream Management Systems (DSMSs) as evidenced by many emerging DSMSs in academia (Aurora [1], STREAM [24], Telegraph [5], Borealis [14]) and industry (Gigascope [12], StreamBASE [25], CMON [26]) etc. However, much of this work applies to a single monitoring site and do not explicitly optimize the communication between multiple sites for continuous monitoring tasks. Some of these methods, especially those that rely on sketches and samples, can be extended to the distributed scenario for on-demand or periodic queries in a communication-efficient manner, but they do not work for continuous aggregate estimation from multiple sites.

Technically, some of the precise problems of our interest here have been studied before in the single site, on-demand queries case. For example, distinct counting has been studied in [15, 16], and we use these primitives to build on. Motivated by sensor networks, simple duplicate-resilient aggregates have been studied in [6, 23]. These begin with computing aggregates (SUM, COUNT) in a duplicate-resilient way, and progress to quantiles and heavy hitters but remain an on-demand computation rather than continuous. In particular, the challenge in our distributed, continuous setting is how to monitor local and global information to decide when to send updated summaries around the network. Our work fits into the model of duplicate-sensitive aggregates proposed in [23], but extends the model to address the continuous nature of query processing not previously studied [6, 23]. Some of our duplicate-resilient aggregates are generalizations of inverse distribution queries, for which solutions were presented in [11], again for on-demand, single site case. We build on all these technical primitives to the continuous, distributed setting by designing what information to maintain at sites, what/how/when to communicate between the sites and the coordinator and apply the resulting protocols to duplicate-resilient aggregates.

Previous works that directly worked in the continuous, distributed setting to optimize the space, time and communication costs as we do here studied non-duplicate resilient aggregates such as heavy hitters [3, 21], quantiles [9], and L_2 sketch maintenance with applications [8]. These papers do not address any problems that are duplicate-resilient. There is a concrete technical challenge we face that is not faced by the papers above because of the duplicate-resiliency. Say there are k monitored sites, and a central site that has the number of distinct elements seen by all the sites thus far. When each of the sites sees a new item, should it update the central site or not? Ideally, it would only send this information if the item has not been observed elsewhere, but sharing this information is costly. Previous work has studied the problem of continuously answering set expression queries [13] by maintaining information about the presence or absence of each item at each site. This approach does not scale when there are very high numbers of items at each site; instead one must keep approximate sketches of the item sets, but it is not clear how to modify this scheme to give guaranteed accuracy using summaries since the protocol relies on tracking frequencies of individual items. Instead, here, we give novel methods, and focus on a variety of aggregates, many of which (quantiles and heavy hitters in particular) cannot be specified as set expressions.

3 Preliminaries

We first formally define the model in which our algorithms will operate, and define the two problems that are at the heart of all our duplicate-resilient aggregate tracking solutions. As in previous work, we acknowledge that communication is delay-prone and messages may get lost, but for the sake of simplicity we focus on the underlying technical problems and so assume that message delivery is instantaneous, i.e. any necessary timestamping and reordering of messages is taken care of. In order to produce communication efficient schemes, we rely on some existing techniques originally proposed in the data streams context. These produce small summaries allowing the estimation of the number of distinct items seen and related functions.

3.1 Distributed Streaming Model

There are k remote sites, each having an insertion update stream S_i, $i = 1 \ldots k$. Each element in S_i is from the integer domain $[U] = \{0 \ldots U-1\}$. Each remote site only communicates with the designated site 0, which acts as a coordinator to answer user queries about the union of all the remote streams. Let $S_0 = S_1 \cup S_2 \cup \ldots \cup S_k$ be the union stream, and N_0 be the number of distinct values in S_0. We let $|S_i|$ denote the number of items (not necessarily distinct) in each stream S_i, and hence $|S_0| = \sum_{i=1}^{k} |S_i|$. Each item v in S_0 has its observed count at site i denoted by $C_{v,i}$, and its overall count $C_{v,0} = \sum_{i=1}^{k} C_{v,i}$.

Our main focus within this model are on two problems, of distinct counting and distinct sampling, which we define formally below. Solutions to these problems are the basis of our methods for tracking duplicate-resilient aggregates.

Definition 1. *The* distinct count problem *is, given parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, for the coordinator site to continuously be able to produce an estimate, DC, so that at any time the current value of DC satisfies*

$$\Pr[N_0(1-\epsilon) \leq DC \leq N_0(1+\epsilon)] > 1-\delta$$

Definition 2. *The* distinct sample problem *is, given parameters T and θ for the coordinator site to continuously be able to produce a sample of pairs $DS = \{(v,c)\}$ of size at most T so that each item v in DS is drawn uniformly from S_0 and its associated count c is a $(1 \pm \theta)$ approximation of $C_{v,0}$, its number of occurrences in S_0.*

In considering our algorithms, we give attention to two important features. *Correctness* says that the algorithm gives the (sometimes probabilistic) guarantee on the answer to the query at the coordinator. *Communication Cost* is (the worst-case bound on) the amount of communication between sites and the coordinator in order to achieve these guarantees. The goal is to minimize the communication cost while guaranteeing correctness. We will also give consideration to the *computational cost* of running the protocol at the remote sites and coordinators, and to the *space cost* of these algorithms, but these are secondary to the goal of minimizing communication. In any event, all our algorithms are relatively fast and space efficient.

3.2 Flajolet-Martin Sketch (FM)

The Flajolet-Martin sketch [15] is a simple, bitmap based algorithm that allows efficient estimation of the number of distinct items. Each entry in the sketch is a bitmap of length $\log U$, where U is an upper bound on the number of distinct items (eg $U = 2^{32}$ or 2^{64}). A hash function h maps items onto the range $[1 \ldots \log U]$ so that the probability of any item mapping onto i is 2^{-i}. For every item seen, v, we set the $h(v)$th bit of the bitmap to 1. After processing all items, the least significant bit in the bitmap that is still 0 is a good estimator for the logarithm of the number of distinct items. To improve accuracy, we can take multiple instances of the same algorithm (using independently chosen hash functions) and repeat, and output the average of the estimates. Under assumptions about the independence of the hash functions, the result is an estimate of the number of distinct items seen, D, that is between $(1-\alpha)D$ and $(1+\alpha)D$ with probability at least $1-\delta$ provided we take at least $O(\frac{1}{\alpha^2}\log 1/\delta)$ repetitions.

An important feature of the data structure that we will use is the ability to merge two FM sketches together to get a summary of the number of distinct items seen over the union of both sets of items. It is a simple observation that the merger is simply the bitwise-or of each pair of corresponding bitmaps. We will use set notations over these sketches Sk: setting $Sk = Sk \cup \{v\}$ denotes adding a new item v to the sketch, and $Sk \cup Sk'$ denotes merging two sketches. We will write $|Sk|$ to denote the estimate of the number of distinct items given by sketch Sk.

3.3 Distinct Sampling

The Distinct Sampling algorithm proposed by Gibbons and Tirthapura [16, 17] is a streaming algorithm that allows the drawing of a uniform sample of items, DS, from the set of distinct items seen in the stream. The method uses a hash function h with the same properties as the FM sketch: the probability of an item hashing to i is exponentially decreasing in i. Based on a level, l, we store each item from the input (and keep count of how many times it is seen) for which $h(v) \geq l$. If the number of items being stored exceeds a threshold, T, then we increment l, and remove from DS any items v with $h(v) < l$. By keeping a total of $T = O(\frac{1}{\alpha^2}\log 1/\delta)$ items sampled from the input, one can answer a variety of queries over the input with high accuracy, as we describe later.

As with the FM sketch, two distinct samples can be easily merged: set l to be the maximum of the ls of the two distinct samples, and merge the set of items. Prune any items which have $h(v) < l$ and sum the counts of items which appear in both samples. If after the merge the size of the set is above the threshold T, increment l and purge items hashing to less than l as usual. The result is precisely the distinct sample that would have been obtained if all items had been processed by a single instance of the algorithm in the centralized model. The challenge is how to to get a good approximation of this at a central site without requiring communication from remote sites every time a new update is received.

4 Distinct Count Tracking

All our algorithms for the Distinct Count problem have the same basic structure: every update that is received at a local site is incorporated into a local sketch Sk_i and the current estimate D_i compared to a threshold. If this local threshold is exceeded, then a communication of Sk_i is triggered between the site and the coordinator. The coordinator updates its data structures Sk_0 and D_0, and sends some information out to some or all sites, such as a copy of its sketch or its current count. To distinguish between the current value of a data structure at one site from a (possibly stale) copy at another site, we use D_i^t, Sk_i^t to denote these values at time t. We usually use this as shorthand to indicate the most recent copy of D_i or Sk_i received by another site or sent out by site i. These commonly used symbols are shown in Figure 1.

Symbol	Description
S_i	Local stream at remote site i
$S_0 = \cup_i S_i$	The union of all streams at the coordinator(conceptual)
N_i, N_0	Current local/conceptual global true number of distinct values
N_i^t, N_0^t	Local/global true number of distinct values at last update
Sk_i, Sk_0	Current local/conceptual global sketch
D_i, D_0	Current local/conceptual global estimate of distinct values from sketch
$C_{v,i}, C_{v,0}$	Local/global count of item v
Sk_i^t, Sk_0^t	Local/global sketch at last update
D_i^t, D_0^t	Local/global estimate at last update
$C_{v,i}^t, C_{v,0}^t$	Local/global count of v at last update

Figure 1: Commonly used symbols

The crucial choices in the design of our solutions are what thresholds to use locally, and what communication the coordinator has with the sites. Our algorithms are designed to be conservative, so that the coordinator can always accurately answer queries. The error at the coordinator comes from two components: the inherent error from using sketches, α, and the maximum permitted "lag", θ. We design our algorithms to try to minimize communication while always communicating to the coordinator when it is possible that the coordinator's answer is no longer accurate.

The basic algorithms are outlined in Figure 2: SKREMOTEUPDATE processes each new item arriving at a remote site i, to update the local sketch and possibly send an update to the coordinator. SKCOORDUPDATE processes the sketches that are received from remote sites. As can be observed, these are essentially quite simple, and the complexity comes in the choice of the two pieces which vary over the algorithms (denoted by underlined names): what threshold to use, $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ (as a function of number of sites k, lag parameter θ and recently sent and received counts); and what messages the coordinator sends back to the site, $\underline{\text{skm}(i, Sk_0)}$. $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ is a function that returns a numerical value, and $\underline{\text{skm}(i, Sk_0)}$ returns a set of pairs $\{(i', m)\}$ meaning that message m is sent to site i'. From our analysis, all our proposed algorithms have the same worst case communication cost [1]; however, experimentally we will see significant variations in this cost on practice, since the constants, and conditions which provoke the worst case behavior, vary across the methods.

Throughout, we discuss the implementation of our algorithms in terms of sending sketches. At the end of this Section we discuss various implementation optimizations,

[1] We omit proofs due to space limitations

```
SkRemoteUpdate($v, i$)
Input: New item $v$ to site $i$
1: $Sk_i \leftarrow Sk_i \cup \{v\}$; $D_i = |Sk_i|$;
2: if $D_i >$ $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ then
3:     send($\underline{i, Sk_i}$) to coordinator
4:     $Sk_i^t = Sk_i$; $D_i^t = |Sk_i^t|$;
```

```
SkCoordUpdate($i, Sk_i$)
Input: Site $i$ sketch $Sk_i$
1: $Sk_0 = Sk_0 \cup Sk_i$
2: $D_0 = |Sk_0|$
3: send($\underline{\text{skm}(i, Sk_0)}$ )
```

Figure 2: Outline of Tracking Algorithms

but for the sake of clarity we pose the algorithms in terms of sending full sketches only.

4.1 Algorithms

We propose four variations of the basic algorithm based on the communications between sites and the coordinator. These are illustrated in Figure 3.

DC Algorithm 1: No Sharing (NS). In our simplest algorithm, the threshold for sending is based solely on information local to the algorithm. So we set $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ = $D_i^t(1 + \frac{\theta}{k})$, and $\underline{\text{skm}(i, Sk_0)}$ = $\emptyset$, meaning no message is ever sent from the coordinator. The advantage of this approach is that it is quite simple. The constraint at site i depends only on local estimates and does not involve coordinating message from the central site. However, for sites with a relatively small number of distinct values, their updates to the coordinator do not affect the answer much, causing unnecessary communication. Subsequent solutions address this issue.

DC Algorithm 2: Shared Count (SC). In our second algorithm, the coordinator ensures that all sites have the current approximate count of distinct items D_0^t on which they can base their threshold. We set $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ = $D_i^t + \frac{\theta}{k} D_0^t$, and $\underline{\text{skm}(i, Sk_0)}$ = $\{(i, D_0) | 1 \le i \le k\}$ is sent to all sites whenever this value changes. This ensures that each site has a larger threshold than in the previous case, and can reduce overall communication cost if the overhead from broadcasting updated counts is not too large.

DC Algorithm 3: Shared Sketch (SS). So far, we have not used much information about other sites. When we receive a new item at some site i, we don't know whether it has been seen before at any of the other sites. To be able to tell this exactly would require very much communication (at least linear in the number of items); however, we can instead ask whether this new item can change the coordinator's sketch. This is a weaker demand, and we can more effectively answer it by sharing the most recent sketch from the coordinator, Sk_0 with the remote sites. Instead of updating a sketch of just their items, they update their local copy of the global sketch, and communicate with the coordinator when the new estimate of the distinct items is above $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ = $D_0^t(1 + \frac{\theta}{k})$. The message sent by the coordinator to all sites is $\underline{\text{skm}(i, Sk_0)}$ = $\{(i', Sk_0) | i' \neq i\}$: the current global sketch is sent to every site except i, which prompted the update (since i can compute this information itself). If an item is seen first at one site, and then later at another site, and the first site communicates to the coordinator in time then the sketch at the second site will not change when the item is seen there. A price is paid by this algorithm: broadcasting a sketch every time there is an update can be very expensive. On some data, the saving from sketch sharing may not be enough to buy back the cost of broadcasting sketches.

DC Algorithm 4: Lazily Shared Sketch (LS). We observe that the main penalty in cost of the Shared Sketch algorithm comes from the eager broadcasting of sketches whenever Sk_0 changes. In some cases there could be relatively little benefit to sending this sketch to everyone. Instead, we can more lazily distribute the updated sketch: every time a site contacts the coordinator, it gets the current Sk_0 in return—sketches are never broadcast. Thus $\underline{\text{skt}(\theta, k, D_0^t, D_i^t)}$ is the same as in the previous algorithm, $D_0^t(1 + \frac{\theta}{k})$, and $\underline{\text{skm}(i, Sk_0)}$ is the single pair $\{(i, Sk_0)\}$: the global sketch is only sent back to site i.

Lemma 1. *All algorithms guarantee error at most $\alpha + \theta$ with probability at least $1 - \delta$. To guarantee error at most ϵN_0, they require communication as follows:*
NS: $O(\frac{k}{\epsilon^3} \log \frac{1}{\delta} \log N_i)$ per site, $O(\frac{k^2}{\epsilon^3} \log \frac{1}{\delta} \log N_0)$ overall; SC: $O(\frac{k}{\epsilon}(k + \frac{1}{\epsilon^2} \log \frac{1}{\delta}) \log N_i)$ per site, $O(\frac{k^2}{\epsilon}(k + \frac{1}{\epsilon^2} \log \frac{1}{\delta}) \log N_0)$ overall; SS and LS: $O(\frac{k^2}{\epsilon^3} \log \frac{1}{\delta} \log N_0)$ overall.

Exact Distinct Count algorithm (EC). For comparison, we give an alternate algorithm that avoids any approximation. Each site looks at each new update, and sends it to the coordinator if it has not been seen before at that site. This is slightly more complicated than the trivial algorithm of sending every update to the coordinator, since it suppresses duplicates: it needs much more space than the previous algorithms, since it has to store the set of seen items exactly, requiring space $\Omega(U)$. The coordinator combines these results and computes N_0 exactly. One can easily see that this algorithm is correct, and the communication cost is $O(\sum_{i=1}^{k} N_i)$. We compare the communication cost of our algorithms to the Exact Count approach.

4.2 Implementation Issues

Communication Optimizations. In implementing these algorithms, there are several natural optimizations to try. For example, the description speaks of sending sketches, but in the initial phases of the algorithm when the number of distinct items seen at each site is relatively low, it would be more efficient to send the set of new items exactly. Making this optimization requires only a relatively small amount of extra space at site to keep the set of items seen so far, since we can switch over to a sketch when this is sufficiently big. Based on this optimization, the total (outward) communication of each remote site using a sketch based algorithm should never exceed that of the Exact Count algorithm described above.

Space and Time Cost. The space cost of our algorithms are much smaller than the space required to record the

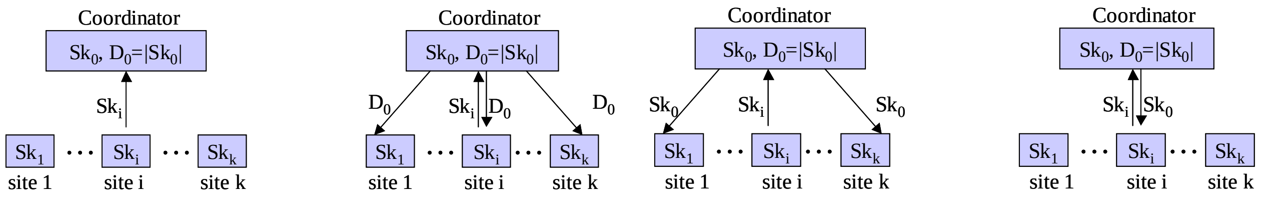


Figure 3: Actions as site i communicates for Distinct Count Tracking algorithms

set of all distinct items seen at a site: to make this trade-off, we need to store at most k/θ times the size of sketch (because we switch over when the number of new items, kD_i/θ is more than the size of a sketch). The space cost of our algorithms at each site is then dominated by this cost, $O(\frac{k}{\alpha^2\theta}\log 1/\delta)$. The remaining space cost is that to store a constant number of sketches and some additional variables. The time to process each update is the time required to update a sketch, and possibly add the item to a set of items seen so far if this is small. If a communication is needed, then a sketch or set of items are sent to the coordinator. In the worst case, the time for this is linear in the size of a sketch, $O(\frac{1}{\epsilon^2}\log 1/\delta)$.

Sketch Type. Although we have couched our discussion in terms of the Flajolet-Martin (FM) sketch data structure, there is little in our algorithms so far that is specific to this structure. Rather, any sketch structure that supports adding new items, merging two sketches and outputting the approximate number of distinct items could be used, eg [7, 4]. Once a particular sketch is chosen, various sketch-specific optimizations could be implemented. For example, we could concisely encode the difference between subsequent sketches sent using compression techniques. We did not include such techniques in our implementation in order to get a clearer comparison between methods.

5 Maintaining Distinct Samples

Our goal is to maintain a distinct sample, DS, at the coordinator with the property that the count of every item in the sample at the coordinator is a good approximation of the item's true count over all sites. As when we maintain sketches, the basic outline of our algorithms are similar, and they vary in two respects: under what threshold the sites update the coordinator with counts of items, and what information the coordinator sends out to the sites.

We simulate running the distinct sample algorithm at the coordinator, so there is a single global notion of l, the "level" of sampling that defines which items we keep. The coordinator keeps information about items whose hash value is at least l, and when the size of this set DS exceeds the threshold T, it increments l and removes all items from DS whose hash value is less than l. It broadcasts the new value of l as soon as l changes: we argue that this is an important step, since it instructs all remote sites that the coordinator is not interested in any items whose hash value

```
DSRemoteUpdate($v, i$)
1: if $h(v) \geq l$ then
2:     $C_{v,i} = C_{v,i} + 1$
3:     if $C_{v,i} > \underline{\text{dst}(\theta, C^t_{v,i}, C^t_{v,0})}$ then
4:         $\Delta_{v,i} = C_{v,i} - C^t_{v,i}$
5:         send($i, v, \Delta_{v,i}$) to coord
```

```
DSCoordUpdate($i, v, \Delta_{v,i}$)
1: $C_{v,0} = C_{v,0} + \Delta_{v,i}$
2: send($\underline{\text{dsm}(i, v, C_{v,0})}$ )
3: if $|DS| > T$ then
4:     $l = l + 1$; broadcast($l$)
5:     for all $v \in DS$ do
6:         if $h(v) < l$ then
7:             delete $v$ from $DS$
```

Figure 4: Distinct Sample Tracking Algorithms

is less than l. This can lead to significant savings, since the fraction of items whose count is l or more is approximately 2^{-l}. Hence we push out changes in l as soon as possible.

In addition to tracking the set of items in level l or higher, the coordinator also tracks the count of items $v \in DS$ as $C_{v,0}$. Keeping exact counts would be too costly, especially in the case when there are a small number of distinct items which all occur very frequently. Instead, we allow approximation in the counts of each item, and study how different approximation strategies affect the communication cost. For each site i and each update v, we update the local count of v as $C_{v,i}$, and determine whether communication with the coordinator is required (based on $\underline{\text{dst}(\theta, C^t_{v,i}, C^t_{v,0})}$). The coordinator maintains the distinct sample, broadcasting a new value of l if necessary, and sending messages $\underline{\text{dsm}(i, v, C_{v,0})}$ concerning item counts. The algorithms are given in outline form in Figure 4.

DS Algorithm 1: Local Counts Only (LCO) Again, we begin by sharing as little information as possible between coordinator and sites. In the Local Count algorithm, we keep information on only the local counts of items v with $h(v) \geq l$. We send whenever $C_{v,i}$ increases by a $(1+\theta)$ fraction, i.e. we set $\underline{\text{dst}(\theta, C^t_{v,i}, C^t_{v,0})} = (1+\theta)C^t_{v,i}$. The coordinator does not need to send any additional information back to sites so we set $\underline{\text{dsm}(i, v, C_{v,0})} = \emptyset$.

DS Algorithm 2: Global Count Sharing (GCS) In the previous algorithm, the use of local counts only means that some sites will send much more often than is needed, since their local counts are much smaller than the global count. For our second algorithm we set $\underline{\text{dst}(\theta, C^t_{v,i}, C^t_{v,0})}$ = $C^t_{v,i} + \frac{\theta}{k}C^t_{v,0}$. We then broadcast the new value of $C_{v,0}$ when it changes, so $\underline{\text{dsm}(i, v, C_{v,0})} = \{(i', v, C_{v,0}) | i' \neq i\}$ (we do not need to send $C_{v,0}$ to i since site i can compute this from its local information).

DS Algorithm 3**: Lazy Count Sharing (LCS)** The broadcast of counts again adds to communication with uncertain benefit. An alternative is to take a lazier approach to count propagation: when a remote site communicates with the coordinator, it receives the current value of $C_{v,0}$ in return. Thus $\underline{\text{dst}(\theta, C^t_{v,i}, C^t_{v,0})}$ is the same as in the previous algorithm, but $\overline{\underline{\text{dsm}(i, v, C_{v,0})}} = \{(i, C_{v,0})\}$.

Lemma 2. *All three algorithms correctly maintain a distinct sample at the coordinator. Counts of items in the sample are correct within a factor of* $1+\theta$. *The worst case communication cost is* $O(\frac{Tk^2 \log |S_0| \log U}{\theta})$.

Exact Distinct Sample algorithm (EDS). To compare the cost of our algorithms, we measure against the algorithm that propagates every update to the coordinator site, which can then draw a sample uniformly from the items in S_0 either directly, or by running Distinct Sampling on the reconstructed stream. This algorithm is clearly correct, and the communication cost is $O(|S_0|)$.

Space and Time Costs. Each site has to keep track of the subset of DS that it has seen at its site, and counts of those items. Since $|DS| \leq T$, the space required is $O(T)$. Each update can be processed quite quickly: we determine whether it is included in T, update the associated count information if so, and communicate with the coordinator if necessary. Thus the time cost is essentially constant (depending on how many different hash functions are used by the distinct sampling).

6 Duplicate-Resilient Aggregates

Using the techniques for the two problems, of tracking distinct counts and distinct samples, we are able to answer many of the duplicate-resilient aggregates described in the introduction. These fall into two main classes, measuring the amount of duplication, and tracking standard (holistic) aggregates in the presence of duplicates. Our results apply to many problems within these two classes, and here we show applications to some of the most important.

6.1 Amount of Duplication

The most basic query, tracking the number of duplicates seen, is answered directly by the Distinct Count itself: if each site sees a stream consisting of events (such as animal interactions, or observed packets), then the number of unique events is given by the distinct count of these events, which we can track accurately up to a $(1 \pm \epsilon)$ factor.

The next query we answer is the number of events that are unique, that is, those that are observed exactly once. Here, we use the distinct sample, and make use of the fact that it samples uniformly from the set of events that are seen, without being biased by the number of times an event is seen. Hence, the number of events that are unique (have count 1) in the distinct sample is a good estimator for the number of events that are globally unique. If the size of the sample, T, is $\Omega(\frac{1}{\epsilon^2} \log 1/\delta)$ then we estimate this quantity up to an error of ϵN_0 with probability at least $1-\delta$.[2].

More generally, once the coordinator has an (approximate) distinct sample, this can be used to answer a variety of queries on the degree of duplication of events. For example, one can estimate the median occurrence count of events that have been seen by computing this function over the sample [11]. The guarantees from using a distinct sample tracked with lag θ is slightly different: in addition to the approximation error due to the size of the sample ($|DS|$), there is additional uncertainty due to θ. For example, when finding the median occurrence count, we return an approximate median (an item whose rank is within $\pm\epsilon N_0$ of the true median) and an approximation of the count (within a $1 \pm \theta$ factor of the true count) with probability at least $1-\delta$, provided $T = \Omega(\frac{1}{\epsilon^2} \log 1/\delta)$.

6.2 Aggregates with Duplicates

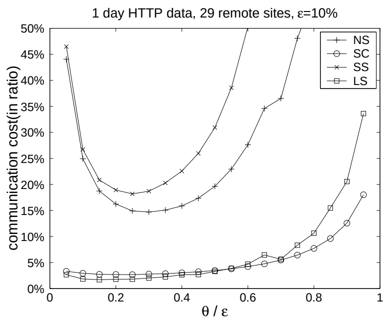
Standard holistic aggregates, such as quantiles and heavy hitters, have been extensively studied in the centralized streaming model without duplicates. Recent work [10, 18] has shown how to adapt sketching techniques in order to find "Distinct Heavy Hitters". This generalizes the heavy hitters problem of finding items which occur frequently to finding items that occur in conjunction with a large number of other items. Formally, the input now consists of streams of *pairs* (v, w), and the goal is, for any v', to estimate the number of distinct pairs $d_{v'}$ in which v' occurs, $|\{(v,w)|(v,w) \in S_0, v = v'\}|$. From this one can find those vs with the largest d_vs etc.

The basic data structure introduced in [10, 18] consists of an array of $c \times d$ FM sketches. Each input pair is by d hash functions $f_1 \ldots f_d$ onto this array: (v, w) is mapped to $f_1(v)$ in the first row, $f_2(v)$ in the second row, and so on. The pair (v, w) is then added to the corresponding FM sketches. It can be shown that taking the minimum of the estimates from each of the sketches that v falls in is a good estimate for d_v [10].

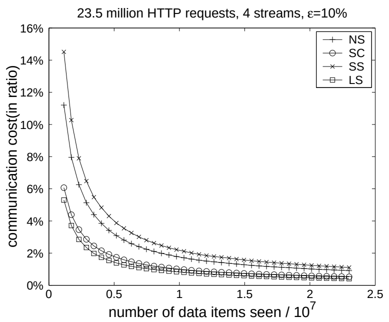
We can apply our distributed sketch tracking algorithm to this data structure: for every update that arrives, we update the d sketches that it affects, and run the sketch tracking algorithms on each sketch independently. This ensures that the coordinator has a good approximation of each sketch, and consequently overall has a good approximation of the whole data structure. The guarantees shown in [10] can then be extended to this distributed scenario in terms of α and θ. One feature shown experimentally in [10] is that the size of the FM sketches needed to give good results are relatively small, so the communication burden should scale reasonably well. [3]

[2]Note that θ does not enter into this expression, because as long as $\theta < 1$ we can distinguish between events that are unique and events that are duplicated.

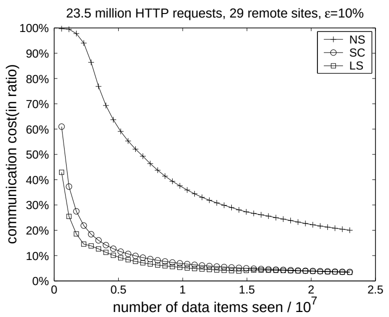
[3] Similar techniques can be applied to tracking quantiles in a duplicate resilient way [10], by again using sketch data structures based on FM sketches; we omit full details due to space limitations.



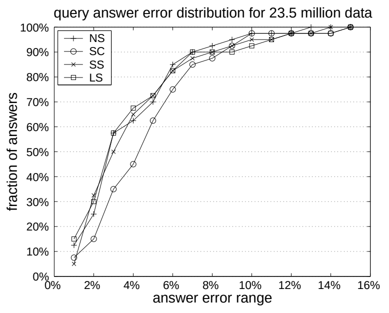
(a) Effect of different θ values, 29 sites



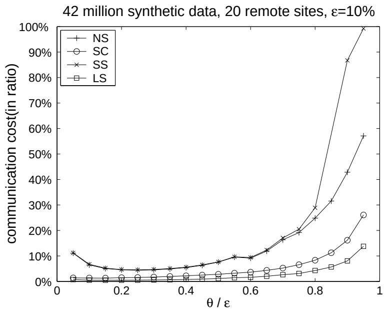
(b) Cost as updates increase, 4 sites



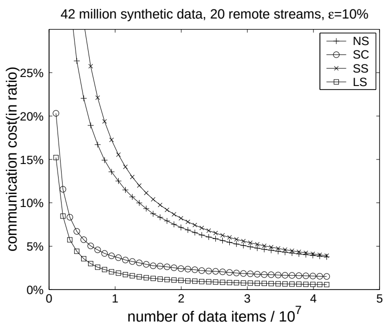
(c) Cost as updates increase, 29 sites



(d) Continuous query accuracy at coordinator



(e) Effect of varying θ



(f) Cost as updates increase, 20 sites

Figure 5: Distinct Counts: (a) – (d) HTTP (clientID, objectID) data from World Cup '98 (e) – (f) Synthetic data.

7 Experiments

Our experimental study focused on the continuous distinct count and distinct sample tracking schemes, their communication costs and query answer accuracy under different parameter settings. We implemented our tracking algorithms in C, with experiments carried out on a 3.2GHz desktop machine. We simulated the remote sites and the coordinator site, and measure the communication cost as the number of bytes sent between the coordinator and each remote site. The communication cost of each algorithm is measured against the cost of the appropriate exact algorithm (EC or EDS), to give a relative cost as a ratio, comparing bytes to bytes. In all experiments, we used the confidence level as 90%, i.e. $\delta = 0.1$.

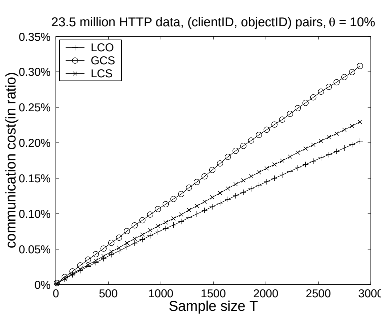
7.1 Data Sets and Methodology

World Cup '98 HTTP request data, obtained from the Internet Traffic Archive [20]. We took entire days of requests, consisting of approximately 20 million tuples. The 29 servers were located in four geographic regions. We experimented with the 29 distinct sites, as well as four sites made by grouping all requests to the same region as a single stream. Each HTTP request record contains clientID, objectID and other attributes. In the experiments presented here, we focus on the clientID attribute, and (objectID, clientID) pairs. Analyzing the data offline found that there are about 16 million distinct (clientID,objectID) pairs and about 120K distinct clientIDs in the data set. Hence, depending on which combination of attributes were used, the degree of duplication varied significantly.

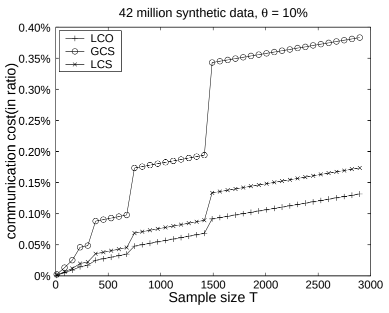
Synthetic data was formed to test the ability of the sites to use information about the distribution seen so far. For each of k sites, we generated a stream in two parts. In the first part each remote stream has n data items, and there are no common data items between two streams. The second part of each stream at each site consists of randomly ordered arrivals of all the kn data items from the first part.

7.2 Experimental Results

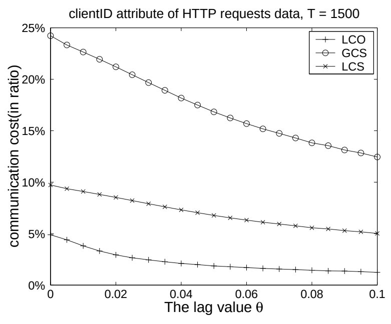
Distinct Count Tracking. Our first set of experiments are on the sketch-based distinct count tracking algorithms. In Figure 5(a) and Figure 5(e) we vary the "lag" value θ while fixing the total error guarantee ϵ at the coordinator. We see savings of between one and two orders of magnitude over the exact algorithm (which in turn can be orders of orders of magnitude cheaper than the naive algorithm of passing every update to the coordinator without any duplicate suppression). When appropriate settings of θ are chosen, the LS (Lazily Shared Sketch) algorithm achieves the least communication cost, edging out the SC (shared count) algorithm. The best setting of θ is $\theta = 0.3\epsilon$ for most algorithms, in line with our worst case analysis; however, for LS, the optimum point appears to be closer to 0.15ϵ. In other experiments (not shown), where we compared the cost in the radio network model (all data is effectively broadcast), we found that the SS algorithm achieved the lowest cost, by a factor of two, indicating that the cost



(a) HTTP data, increasing sample size



(b) synthetic data, increasing sample size



(c) HTTP (clientID) data, increasing θ

Figure 6: Distinct Samples experiments

model is also a significant factor.

From Figure 5(b) (real data with 4 sites), Figure 5(c) (real data with 29 sites, SS not shown because cost is too high) and Figure 5(f) (synthetic data with 20 sites) we can see the communication costs decrease quickly as more updates arrive. LS always gets the lowest cost overall, by an appreciable fraction over its closest competitor SC (seen most clearly on the synthetic data). This shows the value of sharing a sketch, but not eagerly broadcasting it since SS has the worst cost of all methods. In these plots we set θ to be the value that produces optimal communication cost for each algorithm. Testing with other θ values gave similar results. The accuracy of the distinct count is shown in Figure 5(d), where we plot the cumulative distribution of the relative error. We see that our objective, of $\epsilon = 10\%$ relative error $1 - \delta = 90\%$ of the time is met, and is not very high outside this range. Observe that algorithm SC has appreciably lower quality results than the others (a higher line is better).

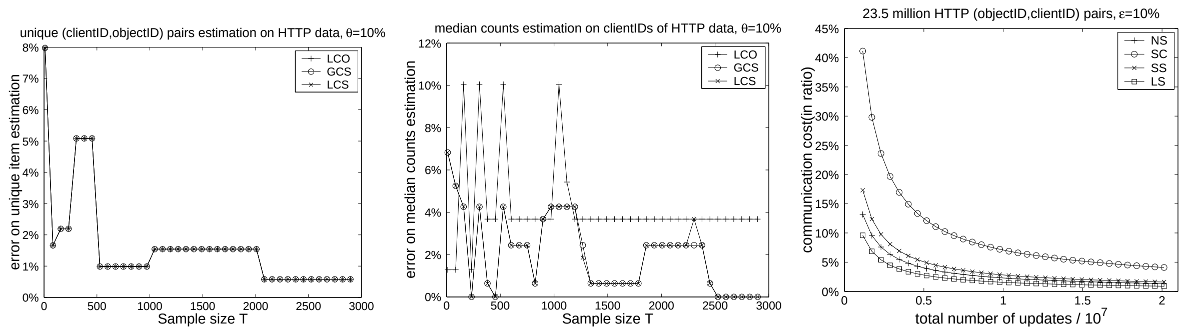
Distinct Sample Tracking. For distinct sample-based algorithms, we first tested on the same two data sets as in previous counting experiments. We fixed the lag value θ and varied the sample size from 10 to around 3000. Figure 6(a) and Figure 6(b) show the communication cost on HTTP (clientID, objectID) pairs and synthetic data. On these data sets (with relatively few duplicates), the communication cost is a saving of two to three orders of magnitude. The communication cost increases linearly with the sample size T, as predicted in the bounds given in Section 5. The LCO algorithm incurs least communication, showing that for sampling, local information gives better results than global. An interesting artifact of the algorithm is visible in Figure 6(b), where sharp discontinuities in the cost are visible. These correspond to points where the Distinct Sampling algorithm terminates at different levels l, which corresponds to an overall cost of approximately twice the previous level.

Figure 6(c) shows the cost (as θ varies) on a data set with much higher levels of duplication, considering only the clientID values in the HTTP data. Here the overall cost is appreciably higher, and the separation of the algorithms clearer. The cost decays weakly as θ increases. We lastly remark on the time cost of our algorithms. For the sketch based algorithms, all methods had similar cost, and our simulator processed in the order of half a million items per second. The distinct sampling algorithms were up to an order of magnitude faster still. This shows that our algorithms should easily cope with the most demanding settings in network traffic monitoring applications.

7.3 Duplicate-Resilient Aggregate Computation

Amount Of Duplication The accuracy of counting the number of unique events has already been discussed in the previous section. For measuring the number of events that are unique, we show an experiment over the HTTP data using the distinct sample to generate estimates. Figure 7(a) shows that a highly accurate result (within 1% error) can be obtained using a sample of size 1000 or so items, which could be collected using communication less than 0.1% of the exact algorithm. All three algorithms achieve the same accuracy here since for this query their estimates are identical (and independent of θ so long as $\theta < 1$). We also compute the median amount of duplication based on the distinct sample. Again, we see high accuracy results in Figure 7(b) with a sample of size a few thousand. Here, we see that there is an appreciable difference in accuracy between LCO and the count sharing algorithms, which are both very similar in terms of accuracy.

Holistic Duplicate-Resilient Aggregates. As outlined in Section 6.2, we can apply our tracking algorithms to tracking more sophisticated aggregates. Here, we focused on the distinct heavy-hitters problem, over (objectID, clientID) pairs: that is, we must identify those objects requested by the largest number of distinct clients, without being influenced by clients requesting the same object multiple times. Figure 7(c) shows the results of processing the streams using a sketch containing about 1500 FM sketches, each of which consisted of 10 repetitions. Here, we observed that the setting of θ had little impact because even at the largest setting of θ, any time a FM sketch changed it would trigger a communication of that FM sketch. We observe that the SC algorithm, which on a single stream was very competitive, is now the worst performer, by a factor of around 4, whereas LS uses the



(a) unique events accuracy (b) median duplications accuracy (c) distinct heavy hitters cost

Figure 7: Duplicate-resilient aggregates

least communication, again by an appreciable factor (about 20% over NS, the closest competitor). The accuracy for this algorithm is very high, with an estimation error of less than 0.1% of the number of distinct pairs for all four methods.

We conclude that, for sketch based methods, there is benefit in sharing sketches in a careful fashion (i.e. LS), especially for problems which build on the basic methods for duplicate resilient aggregates. Knowing a summary of other sites can significantly reduce the cost. In contrast, for distinct sampling, using local information gives the lowest cost. Here, it makes sense to give a bigger threshold to items that have been seen more often at some site.

8 Conclusions

In distributed monitoring applications, the same items may be duplicated at multiple sites, and one needs continuous methods for tracking the extent of duplications and aggregates resilient to them. We formalize such duplicate-resilient event monitoring problems and provide a variety of algorithms that use small space and per-item processing at each site and trade-off the communication between the sites and the central coordinator for accuracy guaranteed. In contrast to the non-duplicate-resilient aggregate tracking studied before where typically there is little or no communication from the coordinator to the sites [8, 9] here, we benefit from the coordinator sharing a summary of the whole data distribution to individual sites. Our results can be extended to handling deletions and sliding window semantics, and a limited set of prediction models in the style of [8, 9][4]. Our experience shows there are many practical challenges in handling the asymmetric information flow and distributed monitoring systems have to optimize this carefully.

References

[1] D. Abadi *et al.* Aurora: a data stream management system. In *ACM SIGMOD*, 2003.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[4] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisian. Counting distinct elements in a data stream. In *RANDOM*, 2002.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: continuous dataflow processing. In *ACM SIGMOD*, 2003.

[6] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *IEEE ICDE*, 2004.

[7] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *VLDB*, 2002.

[8] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[9] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *ACM SIGMOD*, 2005.

[10] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *ACM PODS*, 2005.

[11] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, 2005.

[12] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *ACM SIGMOD*, 2003.

[13] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set cardinality estimation. In *VLDB*, 2004.

[14] Y. Ahmad *et. al.* Distributed operation in the borealis stream processing engine. In *ACM SIGMOD*, 2005.

[15] P. Flajolet and G. N. Martin. Probabilistic counting. In *FOCS*, 1983.

[16] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, 2001.

[17] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, 2001.

[18] M. Hadjieleftheriou, J. W. Byers, and G. Kollios. Robust sketching and aggregation of distributed data streams. BU Technical Report 2005-11, 2005.

[19] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiments with zebranet. In *ASPLOS-X*, 2002.

[20] Internet traffic archive. `http://ita.ee.lbl.gov/`.

[21] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

[22] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM SODA*, 2003.

[23] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys*, 2004.

[24] Stanford stream data manager. `http://www-db.stanford.edu/stream/sqr`.

[25] StreamBase Systems. `http://www.streambase.com`.

[26] K. To, T. Ye, S. Bhattacharyya. CMON: A general-purpose continuous IP backbone traffic analysis platform. Sprint ATL research report RR04-ATL-110309.

[4] Details of these extensions will appear in the full version of this paper.