The Continuous Distributed Monitoring Model*

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

ABSTRACT

In the model of continuous distributed monitoring, a number of observers each see a stream of observations. Their goal is to work together to compute a function of the union of their observations. This can be as simple as counting the total number of observations, or more complex non-linear functions such as tracking the entropy of the induced distribution. Assuming that it is too costly to simply centralize all the observations, it becomes quite challenging to design solutions which provide a good approximation to the current answer, while bounding the communication cost of the observers, and their other resources such as their space usage. This survey introduces this model, and describe a selection results in this setting, from the simple counting problem to a variety of other functions that have been studied.

1. INTRODUCTION

The model of continuous, distributed monitoring is a quite natural one, which arose only in the early years of the 21st century. It abstracts an increasingly common situation: a number of observers are making observations, and wish to work together to compute a function of the combination of all their observations. This abstract description can be applied to a number of settings:

- Network elements within the network of a large ISP are observing local usage of links, and wish to work together to compute functions which determine the overall health of the network.
- Many sensors have been deployed in the field, with the aim of collecting environmental information, and need to cooperate to track global changes in this data.
- A large social network monitors the usage of many compute nodes in data centers spread around the world, and wants to coordinate this information to track shifts in usage patterns and detect any unusual events, possibly indicative of an attack or exploit.

Each of these examples maps naturally onto the outline above: the network elements, sensors and compute nodes respectively play the part of the observers, who want to collaborate in the computation.

There are various "trivial" solutions to these problems. Studying the drawbacks of these helps us to identify the properties to optimize. A first approach is to simply have all the observers send all their observations to a single, centralized location. For cases where the flow of new observations is sufficiently slow, then indeed this is a satisfactory solution. However, in the above scenarios, this places an intolerable burden on the underlying network. For example, in the ISP example, the number of observations may be equivalent to the total number of packets traveling on a link: generating this much extra traffic on the network for the purpose of health monitoring will quickly contribute to the ill-health of the network!

A second approach is to perform "periodic polling": at some fixed interval, say every five minutes, or once an hour, a central monitor polls each observer for information about their observations since the last poll, and collates these together to get a snapshot of the current status. Again, in some situations, this will suffice. Indeed, many network protocols, such as the Simple Network Management Protocol (SNMP) operate on exactly this basis. Still, often this too is insufficient. Firstly, we require that the information needed can be summarized compactly. For example, SNMP allows the reporting of the total amount of traffic (measured in packets or bytes) processed by a network element within a given time window. Quantities like sums and counts of observations, therefore, fit naturally within this setting. However, when the objective is a more complex function, like measuring some non-linear function of all the (distributed) observations, or detecting when some complex event has occurred, it is less clear how periodic polling can operate.

The other limitation of periodic polling is the careful balance needed in setting the frequency of the polling event. Set the gap too narrow, and again the network becomes overloaded with data which may be of limited

*An earlier version of this survey was published in the Proceedings of the Workshop on Algorithms and Models for Distributed Event Processing, 2011

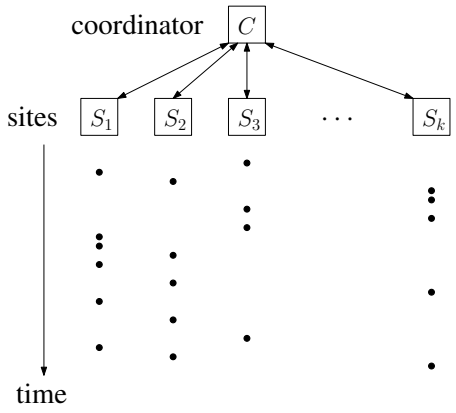

Figure 1: Continuous Distributed Monitoring model

usefulness. But set the gap too large, and the delay between an important event occurring and it being detected by the protocol may become too large.

In continuous distributed monitoring, we aim to address all these concerns. The central idea is to incur minimal communication when there is nothing important being observed, but at the same time to enable rapid (near-instantaneous) updates when necessary.

There has been considerable research effort in this area since its inception. Progress has been made by considering sets of fundamental functions, and describing protocols which provide strong guarantees on the accuracy of the monitoring, while incurring low costs, in the form of communication required, and computational overhead and storage needed by the observers.

Outline. The rest of this survey proceeds as follows. First, we formalize the model, and define the key cost measures. Then we begin by considering a seemingly simple problem in this setting, the problem of counting a fixed number of events, in Section 2. Section 3 considers monitoring the information theoretic concept of entropy, which varies non-monotonically as the number of events increase. We then describe a very general approach to problems in this model via the "geometric approach", in Section 4. Section 5 considers how to maintain a random sample, of either the entire data, or only a recent selection. In Section 6, we outline the history of the model and other results in this area, while Section 7 presents some concluding remarks and open problems.

1.1 Formalizing the model

In total we have k observers (or sites), indexed $S_1, \ldots S_k$. Each observer sees a stream of observations. Typically, each individual observation is quite simple, but in aggregate these define a complex whole. For example, in a communication network, each event might be the arrival of a packet at a router. The description of each event is quite simple: the destination and payload size, say. But the overall distribution of traffic to different destinations observed by multiple routers is very large and complex.

We treat the observations as items $A = a_1, a_2, \ldots a_n$, such that each observation is seen by exactly one observer. There is also a central site, or coordinator, C, who can communicate directly with each observer. For simplicity, we do not allow communication between observers (this can be achieved by sending messages through the coordinator), and we assume each message has unit cost. Varying these assumptions leads to different cost models, some of which are studied in the works described in Section 6. The goal of the monitoring is for the coordinator to continually track some function $f(A)$ over the complete set of observations.

In this survey, we see several different cases of this problem. In 'threshold monitoring', the goal is to determine whether $f(A)$ is above or below a threshold τ. For example, we may want to know when the total network traffic in the last hour exceeds a given amount; or when the entropy of this traffic distribution exceeds a given bound. In 'value monitoring', the goal is to provide an estimate $\hat{f}(A)$ of $f(A)$, such that the difference $|\hat{f}(A) - f(A)|$ is bounded. In the network example, this corresponds to providing an approximate value of the total network traffic; or of the entropy of the traffic distribution. In 'set monitoring', the goal is to provide a set of values which satisfy some property. This could be a uniform sample of the input items, or an approximate top-k (e.g. the top-k most popular destinations in the network).

Figure 1 gives a schematic of the model: communication is between the coordinator and the k different sites. New observations are made over time, which prompts more communication between the parties.

1.2 Comparison to Other Models

There are several other models of computation over data which may be rapidly arriving or distributed. Here, we identify some common models, and outline the key differences.

Communication Complexity. The model of communication complexity focuses on the case where there are two parties, Alice who holds input x and Bob who holds input y, and they wish to work together to compute $f(x, y)$ for some fixed function f [29]. The most important difference between this model and the continuous distributed monitoring case is that the inputs x and y are fixed for communication complexity, whereas in our case, they are allowed to vary. Moreover, it turns out that the main focus of communication complexity is providing lower bounds or impossibility results for various functions, whereas in continuous distributed monitoring, there has been most interest in providing protocols with low communication costs. However, the mod-

els are closely related: techniques from communication complexity have been used to show lower bounds for problems in continuous distributed monitoring [40, 39].

The Data Streaming Model. In the streaming model, a single observer sees a large stream of events, and must keep a sublinear amount of information in order to approximate a desired function f [32]. This omits the key feature of the continuous distributed model, the fact that multiple distributed observers need to compute a function of all their inputs combined. While each observer in our model sees a stream of inputs, the model does not insist that they use sublinear space—rather, the space used by each observer is an additional property of any given protocol. However, it is often desirable that the observers use small space, and techniques from stream processing are therefore useful to help achieve this.

Distributed Computation. Clearly, the continuous distributed model is a special case within the general area of distributed computation. The focus on continually maintaining a function of evolving input distinguishes it from the general case. There are other models within distributed computation, such as the Distributed Streams Model [20, 21] or the Massive, Unordered Data model [17]. These capture the emphasis on distributed streams of data, but focus on a one-time computation, rather than continually tracking a function.

2. THE COUNTDOWN PROBLEM

We begin with a seemingly simple problem which nevertheless admits some fairly sophisticated solutions. In the *countdown problem*, each observer sees some events (non-overlapping, so each event is seen by only one observer), and we wish to determine when a total of τ events have been seen. This is an instance of threshold monitoring. This abstract problem captures many natural settings: we want to raise an alert when more than τ unusual network events have been seen; report when more than 10,000 vehicles have crossed a highway; or identify the 1,000,000th customer; and so on. A trivial solution has each observer send a bit for each event they observe, which uses $O(\tau)$ communication. We aim to considerably improve over this baseline.

A first approach. A smarter approach takes advantage of the fact that there have to be many events at each site before the threshold τ can have been reached. A necessary condition is that at least one of the k sites must observe τ/k events before the threshold can be reached. This leads to a relatively simple scheme (derived from [28]): Each site begins with an initial upper bound value of τ/k, and begins to observe events. Whenever its local count n_i exceeds this upper bound, it informs the coordinator, which collects n_i from each observer, and the n_is are reset to zero. From these, we can determine the current "slack": the difference S between the current count N and the threshold τ, i.e. $S = \tau - N$. This slack can then be redistributed to the observers, so each site now enforces an upper bound of S/k on n_i. Each iteration reduces the slack by a factor of $(1-1/k)$. When the slack (initially τ) reaches k, the observers can switch to reporting every event. The number of slack updates is then

$$\log_{1/(1-\frac{1}{k})}\left(\frac{\tau}{k}\right) = \frac{\log(\frac{\tau}{k})}{\log(\frac{1}{1-\frac{1}{k}})} = O\big(k \log \frac{\tau}{k}\big)$$

The total communication is $O(k^2 \log \tau/k)$, since each update causes communication of $O(k)$.

A quadratic improvement. The step of updating every node whenever one node reports that it has exceeded its current local threshold is somewhat wasteful. This can be improved on by tolerating more updates before a global communication is triggered. This idea was introduced in [10], and we follow the simplified version described in [11].

Now the protocol operates over $\lceil \log(\tau/k) \rceil$ rounds. In the jth round, each observer sends a message to the coordinator when its local count n_i reaches $\lfloor 2^{-j}\tau/k \rfloor$, and then subtracts this amount from n_i. So, in the first round, this bound is $\lfloor \tau/2k \rfloor$. In the jth round, the coordinator waits until it has received k messages, at which point the round is terminated, and the coordinator alerts each site to begin the $j+1$th round, causing the bound to approximately halve. This continues until the bound reaches 1, when each site reports each event when it occurs. Observe now that the communication in each round is more "balanced": the sites send a total of k messages, and the coordinator sends k messages (to inform each site that the new round has begun). Each of these messages can be constant size. Thus, the total communication is $O(k \log \tau/k)$: a factor k improvement over the prior approach.

It also follows immediately that protocol is correct: in any round, the total "unreported" count is at most

$$k\lfloor \tau 2^{-j}/k \rfloor \leq \tau/2^j,$$

while the "reported" count is at most

$$\sum_{i=1}^{j-1} k\lfloor \tau 2^{-i}/k \rfloor \leq \tau \sum_{i=1}^{j-1} 2^{-i} \leq \tau(1-2^{-j}).$$

Hence, the total count never exceeds τ until the final round, when every event is reported directly.

Approximate Countdown. We can improve on the cost of this protocol if we are prepared to tolerate some imprecision in the result. Specifically, we consider protocols which approximate the answer. To approximate, we introduce a parameter ϵ, and ask that the coordinator can determine that the true count is below $(1-\epsilon)\tau$

or above τ; when the true count is in between, then the coordinator can indicate either state.

The protocol is almost identical, but now we terminate when the bound on the unreported count reaches $\epsilon\tau$. The number of rounds is reduced to $\log 1/\epsilon$. This removes τ from the bounds, and makes the total cost of the protocol $O(k \log 1/\epsilon)$ communication.

Countdown lower bounds. We might ask if we can improve further on this result. For deterministic solutions, the answer is no: this bound is tight. This was shown formally in [10]. The intuition is natural: consider the perspective of a single observer, who witnesses a number of events. When this number is substantial enough, it could be part of a global trend, and so must be reported in case they push the total count above the threshold τ. At the same time, it might just be a local phenomenon, in which case any communication does not change the overall answer. Since the observer cannot distinguish these two cases unless it receives a message from the coordinator, then it is forced to communicate. Based on this argument, it is possible to show that the total amount of communication is at least $\Omega(k \log \tau/k)$.

Randomized Countdown Protocol. We can give tighter bounds if we allow both randomization and approximation. Allowing randomization means that we let the protocol have a small probability of giving an erroneous answer at some point in its operation.

The randomized protocol operates as follows, based on a constant c determined by the analysis. Each site observes events, and after collecting a "bundle" $\epsilon^2\tau/(ck)$ of observations, it decides whether to send a message to the coordinator. With probability $1/k$ it sends a message, but with probability $1 - 1/k$, it stays silent. The coordinator declares that enough events have been seen once it has received $c(1/\epsilon^2 - 1/2\epsilon)$ messages. The idea here is that there will be enough opportunities to send messages that with high probability the coordinator will not declare too early or too late. We omit a full analysis here: it can be shown that the amount of communication from sites is $O(1/\epsilon^2)$, and the coordinator is unlikely to declare that the threshold τ has been passed before the true count reaches $(1 - \epsilon)\tau$. Note that this omits the cost to initiate and terminate the protocol, which involves alerting all k sites.

Non-monotonic counts. The approaches outlined for the countdown problem rely critically on the fact that the function being monitored was monotonic: the number of events kept increasing. The non-monotone case is more complex. In general, the count might increase and decrease a lot while close to zero, forcing a lot of communication even for approximate, randomized protocols. However, in cases when there is some randomness in the update streams — for example, when they follow a random walk, or the arrivals are randomly permuted — then stronger guarantees can be provided [31].

3. MONITORING ENTROPY

We next consider monitoring the entropy function from Information Theory. Consider the case where the observers are now witnessing events in the form of arrivals of different items. These arrivals generate an empirical probability distribution (recording the relative proportion of each different item observed), which we can compute the entropy of.

Entropy. Suppose that f_i denotes the number of occurrences of item i observed across the whole system, and m denotes the total number of items (so $m = \sum_i f_i$). Then the empirical probability of i is just f_i/m, and the entropy H of the distribution is given by

$$H = \sum_i \frac{f_i}{m} \log \frac{m}{f_i}$$

The entropy H is an important metric on the distribution: if all f_is are about equal, then the entropy is high, while if most f_is are small and only one is significant, then the entropy is low. It has been argued that changes in entropy are an important indicator of changes in behavior in distributed systems and networks [30].

Entropy Protocol Outline. Arackaparambil *et al.* design a protocol to monitor entropy in the continuous distributed monitoring model [2]. Specifically, they design an approximate protocol, which determines whether the current entropy H is above a given boundary τ, or below $(1 - \epsilon)\tau$. The overall protocol is quite straightforward: the key step is an invocation of an approximate protocol for the countdown problem from Section 2. The protocol proceeds in a number of rounds. In the first round, each site sends every item it receives directly to the coordinator, until some constant number (say, 100) of items have been observed across all sites. This is because the entropy can change quickly in this initial stage. In each subsequent round i, the coordinator computes a parameter τ_i, and runs an instance of the approximate countdown protocol for threshold τ_i, with a constant approximation factor $\epsilon = \frac{1}{2}$. When this protocol terminates, the coordinator contacts each site, which sends a description of its current distribution. The coordinator combines these to estimate the current entropy, and uses this to compute the parameter τ_{i+1} for the next round.

The analysis relies on a basic property of the entropy function: the change in entropy between two points is bounded in terms of the number of new observations. Specifically, if the number of observations at the first point is m, and there are n new arrivals, the change in entropy is at most $\frac{n}{m} \log(2m)$ [2]. Thus, since we know the entropy at the end of round i, and we wish

to know if it changes by at most $\epsilon\tau/2$ (the minimum change needed to change the output of the coordinator), we can set $\tau_{i+1} = \frac{\epsilon\tau m}{2\log(2m)}$, where m is the total number of observations made at the end of round i. Given an upper bound N on the total number of observations, we can ensure that m_i, the total number of observations at the end of round i, satisfies

$$m_{i+1} = m_i + \tau_{i+1} = m_i\left(1 + \frac{\epsilon\tau}{2\log(2m_i)}\right) \geq m_i\left(1 + \frac{\epsilon\tau}{2\log(2N)}\right)$$

and hence the number of rounds to reach N observations is $O(\frac{1}{\epsilon\tau}\log^2 N)$ (provided $\log N \geq \tau\epsilon$).

The communication cost in each round is $O(kD)$, where D is an upper bound on the number of distinct items observed at each of the k sites. When D becomes large, we can instead communicate compact *sketches* of the distribution, which allow us to estimate a function (in this case, entropy) of the combination of the inputs. There are randomized sketches which provide $(1 \pm \epsilon)$ approximation of the entropy using a data structure of size $\tilde{O}(\frac{1}{\epsilon^2})$, where the $\tilde{O}$ notation suppresses logarithmic factors [18, 22].

Lower bounds for entropy monitoring. Lower bounds for this problem can be generating by defining a set of possible inputs chosen so that any individual site cannot tell which case it is in, and so is forced to communicate to resolve this uncertainty. This leads to a deterministic lower bound of $\Omega(k\epsilon^{-1/2}\log(\epsilon N/k))$ and a randomized lower bound of $\Omega(\epsilon^{-1/2}\log(\epsilon N/k))$ [2]. Note that the above protocol is essentially deterministic, and so the stronger bound applies to this case. Recently, Woodruff and Zhang showed stronger lower bounds of $\Omega(k/\epsilon^2)$ for entropy when the input may include arrivals and departures of items [39].

4. THE GEOMETRIC APPROACH

The two results discussed so far considered specific problems (countdown and entropy), and provided tailored protocols based on exploiting specific properties of each function. It is natural to ask whether there are general purpose techniques for generating protocols in this model. The "geometric approach", due to Sharfman, Schuster and Keren aims to do exactly this [35]. The basic idea is to take any desired function, f, and break down the testing of whether $f(x) > \tau$ or $f(x) \leq \tau$ into conditions which can be checked locally, even though x represents the global state of the system. The central result relies on a neat geometric fact, that the area of a convex hull of a set of points can be fully covered by a set of spheres, one sphere incident on each point.

4.1 Formal Description

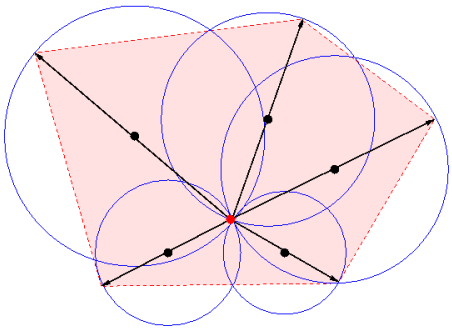

Figure 2: Current estimate e (central red dot), drift vectors Δv_i (arrows out of e), convex hull (dotted outline) and enclosing balls

Preliminaries. Each stream observed at each site is assumed to define a current d dimensional vector v_i. In the countdown case, each v_i was simply the local count; in the entropy case it was the local frequency distribution. With each site we associate a weight λ_i such that these weights sum to 1, i.e. $\sum_{i=1}^k \lambda_i = 1$. These weights might reflect the number of observations at each site, so in this case $\lambda_i = n_i / \sum_{i=1}^k n_i$. Or they may simply be uniform, i.e. $\lambda_i = 1/k$ for all i. Initially, assume that these weights are fixed and known to all nodes.

The weighted combination of all local vectors v_i gives the global vector $v = \sum_{i=1}^k \lambda_i v_i$. The instance of the threshold monitoring problem is then to determine whether $f(v) \leq \tau$ or $f(v) > \tau$, for a fixed function f and threshold τ. For example, we can map the countdown problem into this setting: here, we set $\lambda_i = 1/k$, each v_i is the single dimensional quantity $\langle n_i \rangle$ (number of event observations at site i), and $f(v) = \|v\|_1$. We set τ here to be $1/k$ times the desired threshold. In other words, v is the mean of the event counts at each site, and we want to alert when this mean exceeds a threshold that implies that the total count is above the global threshold.

Protocol Description. At any moment during the protocol, each site has previously informed the coordinator of some prior state of its local vector, v_i'. So the coordinator knows v_i', but not the current state v_i. Based on this knowledge, the coordinator has an estimated global vector $e = \sum_{i=1}^k \lambda_i v_i$. Clearly, if the local vectors v_i move too far from their last reported value v_i', it is possible that the τ threshold may be violated. Therefore, each site monitors its *drift* from its last reported value, as $\Delta v_i = v_i - v_i'$. Thus we can write the current global vector, v, in terms of the current estimate e and the drift vectors:

$$v = \sum_{i=1}^k \lambda_i v_i = \sum_{i=1}^k \lambda_i (e + \Delta v_i) = e + \sum_{i=1}^k \lambda_i \Delta v_i$$

Observe that this is a convex combination of drift vec-

tors. Therefore, the current global vector v is guaranteed to lie somewhere within the convex hull of the drift vectors v_i around e. Figure 2 shows an example in $d = 2$ dimensions, with five drift vectors emanating from an estimate e, and their convex hull. The current value must lie somewhere within this shaded region.

To transform the global condition into a local one, we place a ball on each local drift vector, of radius $\frac{1}{2}\|\Delta v_i\|_2$ and centered at $e + \frac{1}{2}\Delta v_i$. This is illustrated in Figure 2. It can be shown that the union of all these balls entirely covers the convex hull of drift vectors [35]. Thus, we reduce the problem of monitoring the global vector to the local problem of each site monitoring the ball of its drift vector.

Specifically, given the function f, we can partition the space into two sets: X, which is those points x for which $f(x) \leq \tau$, and $\bar{X}$, which is those for which $f(x) > \tau$. The basic protocol is now quite simple: each site monitors its drift vector Δv_i, and checks with each new observation if the ball given by $e + \frac{1}{2}\Delta v_i$ is *monochromatic*, i.e. all points in the ball fall in the same set (X or $\bar{X}$). If this is not the case, then the site communicates to the coordinator. The coordinator then collects the current vectors v_i from each site to compute a new estimate e, which resets all drift vectors to 0. From the above discussion of convex hulls, it is clear that when all balls are monochromatic in the same set (X or $\bar{X}$), then v must also be in the same set, and so the coordinator knows the correct state.

4.2 Extensions to the Geometric Approach

There are several extensions and variations of this basic geometric monitoring scheme which are able to reduce the cost, and avoid some bad cases.

Local Resolution via slack. Whenever a local drift vector creates a non-monotone ball, it causes communication with all sites, to collect their current vectors and distribute the new estimate. This global communication can be be postponed by the coordinator, who can introduce additional "slack", in the form of offset vectors. That is, the coordinator can contact a small number of sites, and allocate a set of vectors δ_i chosen so that the balls for $\Delta v_i + \delta_i$ are now monochromatic, and $\sum_{i=1}^{k} \delta_i = 0$. This idea is discussed in detail in [35]; similar concepts arose earlier, e.g. in work on tracking top-k of frequency distributions [3].

Approximate Thresholds. The version of the protocol described is for an exact version. We can reduce the cost by relaxing this requirement, and introducing an ϵ tolerance around τ. Applying this, when $f(v) < \tau$, we define the sets X and $\bar{X}$ as before, but when we are above the threshold, we define the sets based on $f(x) < (1 - \epsilon)\tau$. This gives more room for the balls to grow, and prevents constant communication when the current value of $f(v)$ is close to τ.

Affine Transformations and Reference Vectors. The use of spherical balls is a natural one, but it is not the only choice. In [36], the authors observe that one can perform any affine transformation on the input, without changing the region covered by the convex hull. In some cases, the resulting ellipsoids can more tightly conform to the convex hull than spheres would. In the same work, the authors discuss replacing the estimate e with a difference reference vector. This can reduce communication by providing a larger "safe area" for the drift vectors to occupy.

Making Predictions. The concept of "prediction" was introduced by Cormode, Garofalakis, Muthukrishnan and Rastogi [9]. The idea is that if items are continually arriving at approximately even rates, then each site can share a simple prediction model of where its distribution will be at any given point in time, rather than relying on a static historical snapshot. Recent work has combined this idea with the geometric approach, and shown that this can be very effective in reducing the cost of monitoring [19].

5. SAMPLING

So far we have concentrated on the case of threshold monitoring: tracking which side of a threshold τ a given function f is on. This is actually quite a general task. For example, we might instead want to monitor the value of f, so that we always have an approximation to its value (value monitoring). But this can be modeled as multiple instances of the threshold monitoring task, for thresholds $1, (1+\epsilon), (1+\epsilon)^2 \ldots$. Tracking all these in parallel can be done by running $O(\frac{1}{\epsilon}\log T)$ instances of the threshold monitoring solution in parallel, where T is maximum value of the function. Although this $1/\epsilon$ factor is large enough to make it worthwhile designing new solutions for value monitoring problems, the techniques and approaches that have been used for value monitoring and threshold monitoring are quite similar.

There are some other monitoring tasks which do not fit either the threshold monitoring or value monitoring paradigms, and instead require us to track the members of a set (set monitoring). For example, we might want to extract information such as which are the k most frequently observed items across all the event streams [3]. In this section, we describe a basic task: to draw a uniform sample from the different event streams, based on the results from [11, 38]. We describe two variations: where we want to sample over all the events ever observed (the infinite window case), and where we want a sample only over the more recent events (the sliding window case).

5.1 Infinite Window

Recall the set-up: we have k distributed sites, each of which is observing events occurring at arbitrary and varying rates. We wish to compute a sample of size s of these events. First, we consider drawing a sample without replacement. The basic idea is to sample across all sites with the same probability p. All sampled items are sent to the coordinator to form a collection, from which s items can be extracted uniformly. Periodically, the coordinator may tell a site to reduce its local p value, and will also prune its collection. We want to bound the resources taken for this process, in terms of the amount of communication, and space needed by the participants.

A simple protocol is as follows [38]: each site i maintains a local p_i, initially 1. For each item that arrives at a site, a random value $0 \leq u \leq 1$ is choosen. If $u \leq p_i$, the item is forwarded to the coordinator, which returns an updated p value to use as the new p_i. The coordinator maintains a set of k items and their corresponding u values, and when a site sends a new item, the coordinator returns the current k'th smallest u value it has seen so far. The correctness of this process follows immediately from the description: the coordinator correctly maintains the k items achieving the k smallest (random) u values across the input, which gives a uniform random sample.

The analysis is a little more involved, but can be done by relating the cost of this simple protocol to a slightly more complex one that keeps a fixed sampling rate p across all sites, which is periodically decreased by a constant factor [11, 38]. The communication cost can then be bounded (with high probability) as $O(k \log_{k/s} n + s \log n)$. One can show a matching lower bound by arguing that this many different items should appear in a random sample over the course of the protocol [11].

The protocol can also be extended to sample with replacement. A trivial solution just runs the above protocol with $s = 1$ in parallel s times over. However, this blows up the costs by a factor of s. Instead, it is possible to take this idea, but to keep all instances of the protocol sampling at the same rate, thus reducing the communication from the coordinator. Analyzing this process allows us to argue that communication of this protocol is bounded by $O((k + s \log s) \log n)$.

5.2 Sliding Windows

A natural variation of continuous distributed monitoring problems is when we do not want to track events across an unbounded history, but rather to see only the impact of recent events. For example, in a network we may only want to include events which have happened within the last hour; in a sensor network, we may only want to track a window of 1 million recent events, and so on. A naive solution would just be to pick a fixed interval—say, 1 hour—and restart the protocol afresh at multiples of this interval. This has the benefit of simplicity, but means that we re-enter a 'start-up' phase every time the protocol restarts, and so we lose information and history around this time. Instead, we describe an approach that is almost as simple as this naive solution, but which provides a sample of an exact sliding window.

A Tale of Two Windows. The key insight needed to generate the solution is due to Braverman, Ostrovsky and Zaniolo [4], who observed that any sliding window can be decomposed into two pieces, relative to a fixed point in time: a growing window as new items arrive after the fixed point, and a shrinking or expiring window of items from before the fixed point. Suppose we want to maintain a sample of items drawn from the last W global arrivals. To draw a sample uniform from these W, we want to take all unexpired sampled items from the expiring window, and make up the shortfall by sampling from those in the growing window. A simple probability calculation shows that this does indeed provide us a uniform sample from the most recent W arrivals.

To implement this idea, we can run an instance of the countdown protocol to count off every W arrivals. We can also run an instance of the above sliding window protocol for drawing a sample beginning at every multiple of W arrivals, which we halt when W further items have arrived. The only additional information needed is that the coordinator needs to know when an item sampled in the expiring window has expired. This can be done by starting a fresh instance of the countdown problem for every sampled item (and terminating this when the item is ejected from the coordinator's collection). This gives the coordinator exactly what is needed to perform the above sampling process: drawing unexpired items from the expiring window, and making up the shortfall from the growing window. The cost of this protocol now grows as $O(ks \log(W/s))$ per window, but this is unavoidable: [11] shows that any protocol for this problem must incur $\Omega(ks \log(W/ks))$ cost.

6. OTHER RELATED WORK

The idea of continuous distributed monitoring is a natural one, and as such it has arisen independently in different areas, under different labels. An early form was as 'Reactive Monitoring' in the networking world. Here, Dilman and Raz introduced a problem that was essentially a variant of the countdown problem, and provided some solutions based on distributing slack amongst the observers [16]. The notion of testing whether a function had exceeded a global threshold appeared under the name of "distributed triggers", and was motivated by Jain *et al.* in a workshop paper [26].

The continuous distributed model has attracted most attention in the data management community. Early work

by Olston, Jiang and Widom focused on tracking a function over single values which could vary up and down, such as monitoring their sum [34]. Here, some uncertainty can be tolerated, so they introduce a natural "filter" approach, which assigns a local filter to each site so that if the current value is within the filter, it does not need to be reported. When a site's value falls outside its filter, the current value is reported, and the filter is re-centered on this value. Over time, some filters can be widened and others narrowed so that the total uncertainty remains bounded, but more slack is allocated to values that are less stable.

A similar approach was used by Babcock and Olston to report the top-k items from a distribution [3]. Again, some tolerance for approximate answers is necessary to avoid communicating every change. The central idea is to choose a set of "adjustment factors" for each item at each site, so that the local distribution after adjustment appears identical to the global distribution. Each site monitors its (adjusted) distribution, and reports if the local (adjusted) top-k changes. In this case, a costly 'rebalancing' stage is invoked.

The use of "predictions" was applied to complex functions such as join sizes (or equivalently, the inner product of large vectors) by Cormode and Garofalakis [7]. Here, the idea was to operate predominantly in "sketch space": a random linear transformation of the input down to low-dimensional vectors. Due to the linearity of the sketch transformation, a prediction based on linear or quadratic growth in different dimensions could be captured by a sketch of the (first order or second order) difference between past values, which in turn is the appropriate difference of sketches. Violations of predictions can be detected by testing the deviation between the actual and predicted sketches.

Huang *et al.* worked on tracking spectral properties of distributed data, where each time step adds a new row to a matrix of observations from different observers. The quantity of interest to be monitored here was the residual energy of the signal after removing the projections along the principal components [25]. Other work studied anomaly detection, where an anomaly occurs when the number of events exceeds an expected rate, over any historical window [24]. This can be seen as a variant of the countdown problem where there is a background process which depletes the number of observed events at a uniform rate. A different approach to this problem is due to Jain *et al.* [27], who consider optimizing slack allocation within a hierarchical network topology, and robustness within a dynamic network (nodes dying, or new nodes joining).

Many other specific functions have been studied in this model, including monitoring the cardinality of set expressions [15] tracking the (large) number of distinct elements observed [12], tracking clusterings of points in a metric space [13], sparse approximation of signals [33], and conditional entropy [1].

The continuous distributed model has also been studied from a more theoretical perspective. [10] revisited various fundamental functions: F_0 (number of distinct elements), F_1 (count/countdown) and F_2 (self-join size or Euclidean norm), and gave the first or improved worst-case bounds for these problems, as well as the first lower bounds. Woodruff and Zhang provided strong lower bounds for a variety of such foundational problems, based on the hardness of a number of primitive problems in communication complexity [39].

Yi and Zhang proposed improved bounds for tracking quantiles and heavy hitters [40]. Specifically, they show how both problems can be solved with total communication $O(k/\epsilon \log n)$ to provide ϵ-approximate results over streams of total length n. Chan *et al.* study the same problems in the context of time-based sliding windows, where only recent events are counted [5]. Cormode and Yi observed that the 'two window' approach used for sampling can also be applied to simplify the analysis, and achieve improved bounds [14].

7. CONCLUDING REMARKS

This survey has aimed to give a flavour of the line of work in continuous distributed monitoring, by highlighting a few problems and approaches, and identifying the breadth of other related work. For a different perspective (with, admittedly, a similar authorial tone), there are surveys and tutorials [8, 6].

Since those prior surveys, there has certainly been progress made in this area. In particular, additional problems have been studied; more robust bounds—both upper and lower bounds—have been proved on the communication costs, as well as other costs such as space; variant models have been introduced, such as sliding windows and the online-tracking model; and a broader set of researchers have worked on related problems (see, for example, the LIFT project, `lift-eu.org/`).

At the same time, many questions posed previously have yet to be fully addressed. Next, I outline two quite different directions for this area that are capable of generating interesting and important results.

Systems for Continuous Distributed Monitoring. While there has been considerable progress on developing protocols and techniques for continuous distributed monitoring, these have yet to translate to practical implementations. There have been several prototype studies of protocols in the works introducing them, which have indicated the potential for orders of magnitude savings in the amount of communication incurred. However, as far as I am aware, these have not translated to widespread adoption of these ideas, or incorporation into standard

protocols. Moreover, these trials have tended to be in simulated environments on recorded data streams, rather than "live" tests. Possibly the lack of uptake of these methods is due to a lack of urgency for the problems considered. While orders of magnitude saving may be possible, if the overhead of centralization, or the delay of polling is considered acceptable, then there is no requirement to implement a more complex monitoring solution. In other words, attention needs to focus on settings where the naive solutions do place an intolerable burden on the network. One interesting example arises in Massively Multi-player Online Role Playing Games (MMORPGs). Here, it has been argued that distributed monitoring of quantities (such as the health scores of players and enemies) would benefit from smarter solutions [23].

In terms of open problems, the basic challenge is to first develop libraries of code, and then evolve these into general purpose systems, so that they can be easily adopted by programmers and data owners. Or, there should exist systems for distributed monitoring which are as accessible and general purpose as traditional centralized database management systems. It remains to determine what classes of functions such tools should support. Should they be based on a collection of "typical" functions (such as the countdown and entropy monitoring problems), or adopt the more generic geometric monitoring approach? Should there be a general purpose, high level query language for flexibly specifying monitoring problems?

A Deeper Theory of Continuous Distributed Monitoring. In recent years, there have been theoretical results shown for problems in continuous distributed monitoring. For the first time, strong upper bounds on the amount of communication of certain protocols have been shown, when previously only heuristic results were known. In some cases these are complemented by lower bounds, sometimes matching or almost matching. Nevertheless, it seems that a richer notion of continuous communication complexity is called for.

There are several powerful results in the literature which could potentially be extended. The famed Slepian-Wolf theorem [37] captures the case where there are correlated sources. They can encode their outputs to allow correct decoding, while using a total amount of communication proportional to the joint entropy. We can cast this "distributed source coding" as a special case of continuous distributed monitoring, where the target is the streams. Then, what we seek is a generalization of Slepian-Wolf, that will capture a function of multiple inputs, rather than just the identity function. This could also take advantage of correlations over time as well as space.

Acknowledgments.

I thank S. Muthukrishnan and Ke Yi for many comments and suggestions.

8. REFERENCES

[1] C. Arackaparambil, S. Bratus, J. Brody, and A. Shubina. Distributed monitoring of conditional entropy for anomaly detection in streams. In *IPDPS Workshops*, 2010.

[2] C. Arackaparambil, J. Brody, and A. Chakrabarti. Functional monitoring without monotonicity. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2009.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[4] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *ACM Principles of Database Systems*, 2009.

[5] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2010.

[6] G. Cormode and M. Garofalakis. Efficient strategies for continuous distributed tracking tasks. *IEEE Data Engineering Bulletin*, 28(1):33–39, March 2005.

[7] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *International Conference on Very Large Data Bases*, 2005.

[8] G. Cormode and M. Garofalakis. Streaming in a connected world: Querying and tracking distributed data streams. In *ACM SIGMOD International Conference on Management of Data*, 2007.

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

[10] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed, functional monitoring. In *ACM-SIAM Symposium on Discrete Algorithms*, 2008.

[11] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *J. ACM*, 59(2):25, 2012.

[12] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate resilient aggregates on data streams. In *IEEE International Conference on Data Engineering*, 2006.

[13] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *IEEE International Conference on Data Engineering*, 2007.

[14] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *ACM Conference on Principles of Distributed Computing (PODC)*, 2011.

[15] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *International Conference on Very Large Data Bases*, 2004.

[16] M. Dilman and D. Raz. Efficient reactive monitoring. In *IEEE INFOCOMM*, 2001.

[17] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. In *ACM-SIAM Symposium on Discrete Algorithms*, 2008.

[18] S. Ganguly and B. Lakshminath. Estimating entropy over data streams. In *European Symposium on Algorithms (ESA)*, 2006.

[19] N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, I. Sharfman, and A. Schuster. Prediction-based geometric monitoring over distributed data streams. In *ACM SIGMOD International Conference on Management of Data*, pages 265–276, 2012.

[20] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 281–290, 2001.

[21] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[22] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *IEEE Conference on Foundations of Computer Science*, 2008.

[23] K. Heffner and G. Malecha. Design and implementation of generalized functional monitoring, 2009. `http://www.people.fas.harvard.edu/~gmalecha/proj/funkymon.pdf`,

[24] L. Huang, M. N. Garofalakis, A. D. Joseph, and N. Taft. Communication-efficient tracking of distributed cumulative triggers. In *ICDCS*, 2007.

[25] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, A. D. Joseph, M. Jordan, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *IEEE INFOCOMM*, 2007.

[26] A. Jain, J. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings of the 3rd Workshop on Hot Topics in Networks (Hotnets)*, 2004.

[27] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula. STAR: Self-tuning aggregation for scalable monitoring. In *International Conference on Very Large Data Bases*, 2007.

[28] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *ACM SIGMOD International Conference on Management of Data*, 2006.

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

[30] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, 2005.

[31] Z. Liu, B. Radunovic, and M. Vojnovic. Continuous distributed counting for non-monotonic streams. In *ACM Principles of Database Systems*, pages 307–318, 2012.

[32] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[33] S. Muthukrishnan. Some algorithmic problems and results in compressed sensing. In *Allerton Conference*, 2006.

[34] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[35] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *ACM SIGMOD International Conference on Management of Data*, 2006.

[36] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *ACM Principles of Database Systems*, 2008.

[37] D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on on Information Theory*, 19:471–480, 1973.

[38] S. Tirthapura and D. P. Woodruff. Optimal random sampling from distributed streams revisited. In *DISC*, 2011.

[39] D. P. Woodruff and Q. Zhang. Tight bounds for distributed functional monitoring. In *ACM Symposium on Theory of Computing*, pages 941–960, 2012.

[40] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *ACM Principles of Database Systems*, 2009.