Efficient Strategies for Continuous Distributed Tracking Tasks

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

Minos Garofalakis
Bell Labs, Lucent Technologies
`minos@bell-labs.com`

Abstract

While traditional databases have focused on single query evaluation in a centralized setting, emerging applications require continuous *tracking of queries on data that is widely* distributed *and constantly updated. We describe such scenarios, and describe the challenges involved in designing communication-efficient protocols for the tracking tasks we define. We outline some solutions to these problems, by abstracting a model of the communication system, defining the tracking tasks of interest, and building query-tracking schemes based on three guiding principles of minimizing global information, using summaries to capture whole data streams, and seeking stability of the protocols.*

1 Introduction

Traditional data-management applications such as managing sales records, transactions, inventory, or facilities typically require database support for a variety of *one-shot queries*, including lookups, sophisticated slice-and-dice operations, data mining tasks, and so on. One-shot means the data processing is essentially done once, in response to the posed query. This has led to an enormously successful industry of database engines optimized for supporting complex, one-shot SQL queries over large amounts of data.

Recent years, however, have witnessed the emergence of a new class of *large-scale event monitoring* applications that pose novel data-management challenges. In one class of applications, monitoring a large-scale system is an operational aspect of maintaining and running the system. As an example, consider the Network Operations Center (NOC) for the IP-backbone network of a large ISP (such as Sprint or AT&T). The NOC has to continuously track patterns of usage levels in order to detect and react to hot spots and floods, failures of links or protocols, intrusions, and attacks. A similar example is that of data centers and web-content companies (such as Akamai) that have to monitor access to thousands of web-caching nodes and do sophisticated load balancing, not only for better performance but also to protect against failures. Similar issues arise for utility companies such as electricity suppliers that need to monitor the power grid and customer usage. A different class of applications is one in which monitoring is the goal in itself, such as wireless sensor networks which monitor the distribution of measurements for trend analysis, detecting moving objects, intrusions, or other adverse events.

Examining these monitoring applications in detail allows us to abstract a number of common elements. Primarily, monitoring is *continuous*, that is, we need real-time tracking of measurements or events, not merely one-shot responses to sporadically posed queries. Second, monitoring is inherently *distributed*, that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

information through a communication network. This also means that there typically are important *communication constraints* owing to network-capacity restrictions (e.g., in IP-network monitoring, where the collected utilization and traffic is very voluminous [6]) or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [14]). Furthermore, each remote site may see a *high-speed stream* of data and has its own local resource constraints, such as *storage-space* or *CPU-time* constraints. This is true for IP routers that cannot possibly store the log of all observed traffic due to the ultra-fast rates at which packets are forwarded. This is also true for the wireless sensor nodes, even though they may not observe large data volumes, since they typically have very small memory onboard.

In addition, there are two key aspects of such large-scale monitoring problems. First, one needs a way to effectively track the *complete distribution of data* (e.g., IP traffic or sensor measurements) observed over the collection of remote sites, as well as *complex queries (e.g., joins)* that combine/correlate information across sites. The ability to have an accurate picture of the overall data distribution and effectively correlate information across remote sites is crucial in understanding system behavior and characteristics, tracking important trends and correlations, and making informed judgments about measurement or utilization patterns. In other words, while hardwired outlier detection methods can be of use for certain applications (e.g., network anomaly detection), data-distribution and correlation information gives us a much broader and more robust indicator of overall system behavior — such indicators are critical, for instance, in network-provisioning systems that try to provision routing paths with guaranteed Quality-of-Service parameters over an IP network (e.g., delay or jitter for a VoIP application). Second, answers that are precise to the last decimal are typically not needed when tracking statistical properties of large-scale systems; instead, *approximate estimates* (with reasonable guarantees on the approximation error) are often sufficient, since we are typically looking for indicators or patterns, rather than precisely-defined events. Obviously, this can work in our favor, allowing us to effectively tradeoff efficiency with approximation quality. To summarize, our focus is on large-scale monitoring problems that aim to continuously provide accurate summaries of the complete data distribution and approximate answers to complex queries over a collection of remote data streams. Solutions for such monitoring problems have to work in a distributed setting (i.e., over a communication network), be real-time or *continuous*, and be space and communication efficient; furthermore, approximate, yet accurate, answers suffice.

2 Challenges

In this section, we outline some of the key challenges to address in the area of approximate distributed data-stream tracking. The remainder of the paper further expands on these challenges, and discusses our technical approach and some of our recent results in this area.

• **To Develop New Models and System Architectures for Tracking Distributed Stream Queries.** New frameworks and query-processing architectures are required for the effective, approximate tracking of massive streams in a distributed setting, with guaranteed upper bounds on the approximation error. A suite of architectures may prove suitable, including *single-level hierarchies* (where a single central processing system directly communicates with and coordinates a collection of distributed monitor sites), to *multi-level hierarchies*, and the most general *fully-distributed architectures* (where no central coordinator exists and the goal is for all the distributed monitor sites to efficiently reach consensus on the approximate answer of a distributed stream query). The most applicable from the IP network monitoring point of view is the single-level hierarchy, although other distributed data streaming applications may find the other architectures most suitable.

• **To Identify Key Distributed Data Stream Monitoring Problems.** It is necessary to isolate the basic problems, drawing intuition from multiple application scenarios (such as IP traffic monitoring, mining message streams, and analysis of sensor measurement data), consultation with experts (such as network operators), and so on. Concrete examples that have already begun to be studied include: tracking top-k items [3]; set expression

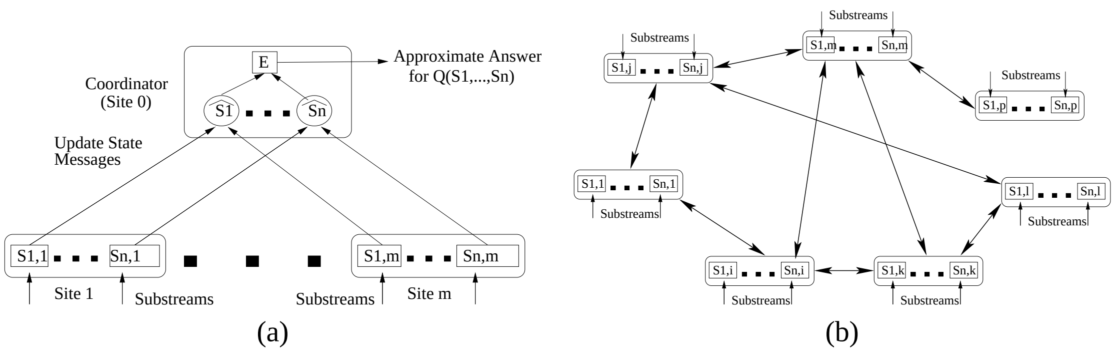

Figure 1: (a) Single-Level Hierarchical Stream-Processing Model. (b) Fully-Distributed Model.

cardinality [7]; histograms and quantiles [5]; and join query estimation [4].

• **To Develop New Algorithmic Techniques and Lower Bounds.** In designing novel, communication-efficient algorithms for tracking distributed-stream queries, a key step is to draw on known approximation methods in the centralized data-stream context that are time and space efficient (e.g., based on the linear-projection sketches of [1, 2, 9], or the hash-sketches of [10, 11]). The end goal is overall solutions that are simultaneously efficient in time, space, and communication, while trading off result accuracy. A complementary algorithmic challenge is to prove stronger lower bounds for these problems, which measure the required communication to *continuously* track quantities of interest and show hard theoretical limits in worst-case scenarios.

3 System Models

In this section, we outline a formulation of appropriate models for continuous distributed tracking tasks. Figure 1(a) depicts the *Single-level hierarchical architecture* for update-stream processing, with $m + 1$ sites and n (distributed) update streams. Stream updates arrive continuously at *remote* sites $1, \dots, m$, whereas site 0 is a special *coordinator* site that is responsible for generating answers to continuous user queries Q over the n distributed streams. In this simple hierarchical model, the m remote sites do not communicate with each other; instead, as illustrated in Figure 1(a), each remote site exchanges messages only with the coordinator, providing it with state information for (sub)streams observed locally at the site. Note that such a hierarchical processing model is, in fact, representative of a large class of applications, including network monitoring where a central Network Operations Center (NOC) is responsible for processing network traffic statistics (e.g., link bandwidth utilization, IP source-destination byte counts) collected at switches, routers, and/or Element Management Systems (EMSs) distributed across the network.

At each remote site j, the n update streams render n distinct multi-sets $S_{1,j}, \dots, S_{n,j}$ of elements, that are assumed (without loss of generality) to lie in the integer domain $[M] = \{0, \dots, M-1\}$. Each stream update at remote site j is a triple of the form $< i, e, \pm v >$, where i identifies the multi-set $S_{i,j}$ being updated, $e \in [M]$ is the specific data element whose frequency changes, and $\pm v$ is the net change in the frequency of e in $S_{i,j}$, i.e., "$+v$" ("$-v$") denotes v insertions (resp., deletions) of element e. [1] For each $i = 1, \dots, n$, let $S_i = \cup_j S_{i,j}$. Thus, S_i reflects the global state of the i^{th} update stream, while each multi-set $S_{i,j}$ captures the local state of stream i at site j. In the development that follows, we will loosely refer to S_i and $S_{i,j}$ as streams even though the intended meaning is the current states of the underlying streams.

Multi-level hierarchical system architectures have individual distributed substream-monitoring sites at the

[1]Other relevant update models are to consider only transactions that are recent (sliding-window model), or to forbid deletions (insertions-only model).

leaves and internal nodes of a general *communication tree,* and the goal is to effectively track a continuous stream query $Q(S_1, \ldots, S_n)$ at the *root node* of the tree. The most general setting are *fully-distributed system architectures*, where individual monitor sites are connected through an arbitrary underlying *communication network* (Figure 1(b)); this is a distinctly different distributed system architecture since, unlike hierarchical systems, no centralized authority/coordination exists and the end goal is for all the distributed monitors to efficiently reach some form of *consensus* on the approximate answer of a distributed stream query. Such generalized distributed stream-processing architectures are representative of, e.g., wide-area networks and sensornet environments. Sensornets, for instance, typically process queries *in-network*, with the sensors performing local measurements and processing, and communicating partial results either over a dynamically-maintained *routing tree* towards a powerful root basestation [12, 13, 14], or through *fully-decentralized, broadcast-based* communication protocols.

It is also important to consider the characteristics of the underlying distributed computing infrastructure; for example, to distinguish between "fat" versus "thin" monitoring clients (e.g., NOCs for local-area networks versus switch Element Management Systems (EMSs), or heavy versus tiny sensornet motes).

4 Distributed Stream Processing Problems

Different distributed stream monitoring tasks rely on a variety of different queries to be evaluated. In this section, we describe some of the specific problems that we have identified.

Tracking Set-Expression Cardinality. The problem of *estimating the result cardinalities of set expressions* over the distributed streams is, given a set expression E over the streams $S_1, \ldots, S_n$ (with the standard set operators $\cup, \cap$, and $-$ as connectives), to estimate the answer to the query $Q = |E|$, i.e., the number of distinct elements in E. For example, the problem of counting the number of distinct IP source addresses described in the context of Distributed Denial of Service (DDoS) monitoring, is a special case of the more general set-expression cardinality estimation problem. As another example of a DDoS-monitoring scenario, we may want to employ the set difference cardinality query $|S_{curr} - S_{prev}|$ to detect significant traffic deviations in real time – here, S_{curr} is the set of source-IP addresses for the sliding window spanning this past week (until now), and S_{prev} denotes the set of source-IP addresses from the week prior to that (e.g., two weeks ago). Set-expression cardinality queries can also provide valuable information to an ISP for network traffic engineering. For instance, suppose that S and T are the sets of IP destination addresses observed in the flows from two of the ISP's peers. Then, the set intersection query $|S \cap T|$ yields the overlap between the destination sets for IP traffic that the peers transmit over the ISP's network. In order to increase network efficiency, it makes sense to locate the traffic exchange points with peers for which this overlap is large, at the same set of border routers (that are optimally situated to carry traffic for the common destinations). A solution based on a dynamic-charging scheme with costs charged to each member of a set, and global broadcast when these charges are recalibrated, which shows significant cost savings over sending every update is given in [7].

Tracking Distributed Stream Summaries. Another important distributed stream processing task is that of continuously *summarizing the frequency distribution* of a distributed stream of updates S_i, comprising one or multiple data attributes. *Quantiles, Histograms* and *wavelet-coeffi cient synopses* are natural and very useful methods for summarizing data distributions for the purposes of data visualization and analysis that have been extensively studied in the database literature. The basic goal of all these earlier efforts has been, given a multi-attribute, disk-resident data set and a fixed amount of space, construct the "best" histogram or wavelet synopsis for the given space budget; that is, build the synopsis that minimizes an overall error metric in the approximation of the data distribution, for some suitably-defined notion of error. More recent work has also addressed the problems of quantile finding, histogram and wavelet construction over a single, continuous stream of data in a centralized context.

Tracking Distributed Stream Joins. SQL *join queries* represent another fundamental way of correlating information from two or more distinct data sets (or, streams); thus, effective support for such queries is very important for most of our target distributed stream processing application domains, including network management and sensornet query processing. This class of queries takes the general form $Q =$ "SELECT AGG FROM $S_1, S_2, \ldots, S_n$ WHERE $\mathcal{E}$", where AGG is an arbitrary aggregate operator (e.g., COUNT, SUM, or AVERAGE) and $\mathcal{E}$ represents the conjunction of of equi-join constraints of the form $S_i.A_j = S_k.A_l$ ($S_i.A_j$ denotes the j^{th} attribute of the streaming relation S_i).

5 A General Query-Tracking Framework

We now outline some of our recent work and results on developing general query tracking solutions in the single-level hierarchical model. We have applied this approach to both the quantile and join tracking problems [4, 5].

The goal of our tracking algorithms is to ensure strong error guarantees for approximate answers to queries at the coordinator over the collection of global streams S_i while minimizing the amount of communication with the remote sites. We can also identify other important design desiderata that helped guide our overall approach: (1) *Minimal global information exchanges* — schemes in which the coordinator distributes information on the *global* streams to remote sites would typically need to re-broadcast up-to-date global information to sites (either periodically or during some "global resolution" stage [3, 7]) to ensure correctness; instead, our solutions are designed to explicitly avoid such expensive "global synchronization" steps; (2) *Summary-based information exchange* — rather than shipping complete update streams $S_{i,j}$ to the coordinator, remote sites only communicate concise summary information (e.g., sketches, quantiles) on their locally-observed updates; and, (3) *Stability* — intuitively, the stability property means that, provided the behavior of the local streams at remote sites remains reasonably stable (or, *predictable*), there is no need for communication between the remote sites and the coordinator.

Our schemes avoid global information exchange entirely by each individual remote site j continuously monitoring only properties of their own, *local* update streams $S_{i,j}$. When a certain amount of change is observed locally, then a site may send a concise *state-update* message in order to update the coordinator with more recent information about its local update stream, and then resumes monitoring its local updates (Figure 1(a)). Such state-update messages typically comprise a small summary of the offending local stream(s) (along with, possibly, additional trend information), to allow the coordinator to continuously maintain accurate approximate answers to user queries. The tracking scheme depends on two parameters ϵ and θ, where: ϵ captures the error of the local summaries communicated to the coordinator; and θ captures (an upper bound on) the deviation of the local-stream properties at each remote site involved in the query since the last communication with the coordinator. The overall error guarantee provided at the coordinator is given by a function $g(\epsilon, \theta)$, depending on the specific form of the query being tracked. Intuitively, larger θ values allow for larger local deviations since the last communication and, so, imply fewer communications to the coordinator. But, for a given error tolerance, the size of the ϵ-approximate summaries sent during each communication is larger (since $g(\epsilon, \theta)$ is always increasing in both parameters). This tradeoff can be analyzed under worst-case assumptions, and an optimal setting of ϵ and θ can be given, for an overall error bound and a fixed $g(\epsilon, \theta)$.

A local summary communicated to the coordinator gives an (ϵ-approximate) snapshot of the $S_{i,j}$ stream at time t.[2] To achieve *stability*, a crucial component of our solutions are concise *summary-prediction models* that may be communicated from remote sites to the coordinator (along with the local stream summaries) in an attempt to accurately capture the anticipated behavior of local streams. The key idea here is to enable each site j and the coordinator to share a prediction of how the stream $S_{i,j}$ evolves over time. The coordinator employs this prediction to answer user queries, while the remote site checks that the prediction is close (within θ bounds) to

[2]To simplify the exposition, we assume that communications with the coordinator are instantaneous. In the case of non-trivial delays in the underlying communication network, techniques based on time-stamping and message serialization can be employed to ensure correctness, as in [15].

the actual observed distribution $S_{i,j}$. As long as the prediction accurately captures the local update behavior at the remote site, no communication is needed. To keep communication costs low, we need to reflect the predicted distribution with a predicted summary, by taking advantage of linearity or other properties of the summaries. For example, the random sketches of [1, 2] are linear projections of the distribution, so scalings and combinations of the distribution can be predicted by scaling sketches; heavy-hitter counts can be predicted by scaling previous counts; quantiles may be assumed to remain the same, or change slowly; and so on. Thus, our predictions are also based solely on concise summary information that can be efficiently exchanged between remote site and coordinator when the model is changed. The key insight from our results is that, as long as local constraints are satisfied, the combined predicted summaries at the coordinator are basically equivalent to *$g(\epsilon, \theta)$-approximate summaries* of the global data streams. We now describe how we have mapped this general outline onto two distinct problems, and the different results that follow from fleshing out the required details of our framework.

Quantile Tracking. For quantile tracking, the summaries used are the ϵ-quantiles of each remote stream [5]. It can be shown (Theorem 3.1 of [5]) that, so long as the deviation between the predicted ranks of the most recently recorded set of ϵ-quantiles observed locally and their true ranks is less than $\theta|S_{i,j}|$, the coordinator can accurately answer quantile queries with bounded error. The error function, $g(\epsilon, \theta)$ is given by $\epsilon + \theta$. In order to achieve stability, a variety of prediction models can be applied. The simplest is to assume that ranks are fixed; the effect of this is to cause a communication when the number of updates (in an insert-only model) exceeds a θ-fraction of the size of the stream at the last update, $|S_i|$. Analyzing this shows that the communication cost is optimized by setting $\theta = \epsilon$, and the overall cost is $O(\frac{1}{\epsilon^2} \ln |S_{i,j}|)$ (Lemma 3.4 of [5]). A more sophisticated prediction tracks the rate of change of each local stream, and models this with a linear "velocity" component. Although not easily amenable to analysis, this model shows significant communication savings over the naive solution of sending every update to the coordinator [5].

Join Size Estimation (Sketch Tracking). To track binary and multi-way join aggregates between streams (and a variety of other problems) [4], the summaries used are variations of the sketches defined by Alon et al. [1]. If the coordinator holds a sufficiently accurate sketch of each local stream, then it can be shown (Theorem 3.1 of [4]) that the coordinator can build a sketch that is $g(\epsilon, \theta)$-accurate for answering join-size queries. Formally, the answer has additive error $g(\epsilon, \theta) = \epsilon + (1+\epsilon)^2((1+\theta)^2 - 1) \approx \epsilon + 2\theta$ times the product of the L_2 norms of the frequency vectors (which is of comparable magnitude to the error due to using approximate sketches). A prediction model is applied to predict the distribution at time t, but, because of the linearity properties of the sketches, the effect is to produce a *predicted sketch*. In order to give this guarantee, each remote site ensures that the deviation between the sketch of the true distribution and its predicted sketch is bounded by $\theta \|S_{i,j}\|_2$, a θ-factor times the L_2 norm of the stream. Again, simple prediction models based on the assumption that the distributions generated by the streams remain static are reasonably successful at reducing the amount of communication between remote sites and the coordinator. More sophisticated prediction models capture the dynamics of streams by modeling first- and second-order effects: the prediction is made up by adding a "velocity" and "acceleration" component to the last sketch sent to the coordinator. This is often very successful at capturing the movement of the streams, hence further reducing communication cost, but requires a careful balancing act: since the coordinator bases its approximate answers on the current prediction, details of the model must be sent to the coordinator. Thus, a more complex model also requires more communication, and the correct balance between complexity of the model and accuracy with which it captures the stream distributions needs to be struck.

The overall result of these approaches are conceptually simple but effective schemes for tracking a variety of aggregates of interest. In practical evaluations [4, 5] these can be seen to use an amount of communication that is significantly smaller (tens or even hundreds of times less) than the cost of shipping every update to the central coordinator for processing, while giving the overall $g(\epsilon, \theta)$ accuracy guarantees for query answering.

Generalizing the above results gives solutions for other query types. The quantile structure naturally extends to finding the "heavy hitters" (frequent items), by applying similar models and bounds to the local heavy-hitter items, and combining them appropriately at the coordinator [5]. More generally, our sketch-tracking solution gives a $g(\epsilon, \theta)$-approximate sketch at the coordinator; this can then be used to answer a variety of data-analysis problems which make use of sketches: self-join size approximation, building multi-dimensional histograms and wavelets, answering point and range queries, and so on [4].

Our techniques also extend naturally to the multi-level hierarchical setting, since they effectively give a $g(\epsilon, \theta)$-approximate summary (quantiles or sketch) at the coordinator. Therefore, if the coordinator runs the same protocol with its parent in the tree, the result at the next level is a $g(g(\epsilon, \theta), \theta)$-approximate summary. This approach can be continued up the hierarchy, increasing the error bound at each step [5]. However, many challenges remain unanswered, including (a) how to better take advantage of correlations or anti-correlations across remote sites (the schemes described in this section treat each remote site entirely independently, which gives simple to manage solutions, and no global information, but could benefit from utilizing greater knowledge about the behavior of the streams); and, (b) how to extend to the more general, fully-distributed model. There is much scope for work that builds very different models of the update streams—a completely different notion of models is used in [8] for choosing which sensors (sites) to query in a one-shot query evaluation setting.

6 Concluding Remarks

We have introduced a class of problems based on continuous, distributed tracking tasks. We have described three key challenges: to develop new models and architectures for such queries; to identify key problems in this area; and to provide new algorithms and theoretical lower bounds for them. We have also outlined some of our recent and ongoing work to address these three challenges.

Acknowledgments. We thank S. Muthukrishnan for several useful discussions related to this work.

References

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

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

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

[4] G. Cormode and M. Garofalakis. "Sketching Streams Through the Net: Distributed Approximate Query Tracking". Bell Labs Tech. Memorandum, 2005.

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

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

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

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

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

[10] P. Flajolet and G. N. Martin. "Probabilistic Counting Algorithms for Data Base Applications". *Journal of Computer and Systems Sciences*, 31:182–209, 1985.

[11] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Set Expressions over Continuous Update Streams". In *ACM SIGMOD*, 2003.

[12] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. "Beyond Average: Towards Sophisticated Sensing with Queries". In *IPSN*, 2003.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks". In *OSDI*, 2002.

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

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