Distributed Data Streams

Minos Garofalakis
Yahoo! Research and Univ. of California, Berkeley
minos@acm.org

SYNONYMS

None.

DEFINITION

A majority of today's data is constantly evolving and fundamentally distributed in nature. Data for almost any large-scale data-management task is continuously collected over a wide area, and at a much greater rate than ever before. Compared to traditional, centralized stream processing, querying such large-scale, evolving data collections poses new challenges, due mainly to the physical distribution of the streaming data and the communication constraints of the underlying network. Distributed stream processing algorithms should guarantee efficiency not only in terms of *space* and *processing time* (as conventional streaming techniques), but also in terms of the *communication load* imposed on the network infrastructure.

HISTORICAL BACKGROUND

The prevailing paradigm in database systems has been understanding the management of *centralized* data: how to organize, index, access, and query data that is held centrally on a single machine or a small number of closely linked machines. Work on parallel and distributed databases has focused on different notions of consistency and methods for effectively distributing query execution plans over multi-node architectures — the issues of monitoring or querying distributed, high-speed data streams in a space-, time- and communication-efficient manner were not addressed in this realm. Similarly, the bulk of early research on data-streaming algorithms and systems has concentrated on a *centralized* model of computation, where the stream-processing engine has direct access to all the streaming data records. Centralized stream-processing models can obviously ignore communication-efficiency issues; still, such models are also painfully inadequate for many of the prototypical data-streaming applications, including IP-network and sensornet monitoring.

SCIENTIFIC FUNDAMENTALS

Tracking and querying large-scale, evolving data collections poses a number of challenges. First, in contrast with conventional, centralized models of data-stream processing, the task is inherently *distributed*; that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange information through a communication network. This also means that there typically are important *communication constraints* owing to either network-capacity restrictions (e.g., in IP-network monitoring, where the volumes of collected utilization and traffic data can be huge [7]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [18]). Second, each remote site may see a *high-speed stream* of data and, thus, must solve a local (centralized) stream-processing problem within its own local resource limitations, such as *space* or *CPU-time* constraints. This is certainly true for IP routers (that cannot possibly store the log of all observed packet traffic at high network speeds), as well as wireless sensor nodes (that, even though may not observe large data volumes, typically have very little memory on-board). Finally, applications often require *continuous monitoring* of the underlying streams (i.e., real-time tracking of measurements or events), not merely one-shot responses to sporadic queries.

To summarize, the focus is on techniques for processing queries over collections of remote data streams. Such techniques have to work in a distributed setting (i.e., over a communication network), support one-shot or continuous query answers, and be space, time, and communication efficient. It is important to note that, for most realistic distributed streaming applications, the naive solution of collecting all the data in a single location is simply *not* a viable option: the volume of data collection is too high, and the capacity for data communication relatively low. Thus, it becomes critical to exploit local processing resources to effectively minimize the burden on the communication network. This establishes the fundamental concept of "*in-network processing*": if more computational work can be done *within* the network to reduce the communication needed, then it is possible to significantly improve the value of the network, by increasing its useful life and communication capacity, and extending the range of computations possible over the network. This is a key idea that permeates the bulk of existing work on distributed data-stream processing — this work can, in general, be characterized along three (largely orthogonal) axes:

(1) *Querying Model:* There are two broad classes of approaches to in-network query processing, by analogy to types of queries in traditional DBMSs. In the *one-shot* model, a query is issued by a user at some site, and must be answered by "pulling" the

current state of data in the network. For simple aggregates, this can be done in a few rounds of communication where only small, partial-aggregate messages are exchanged over a spanning tree of the network. For more complex, *holistic* aggregates (that depend on the complete data distribution, such as quantiles, topk-k, count-distinct, and so on), simple combination of partial results is insufficient, and instead clever composable summaries give a compact way to accurately approximate query answers.

In the *continuous* model, users can register a query with the requirement that the answer be tracked continuously. For instance, a special case of such a continuous query is a *distributed trigger* that must fire in (near) real-time when an aggregate condition over a collection of distributed streams is satisfied (e.g., to catch anomalies, SLA violations, or DDoS attacks in an ISP network). This continuous monitoring requirement raises further challenges, since, even using tree computation and summarization, it is still too expensive to communicate every time new data is received by one of the remote sites. Instead, work on continuous distributed streams has focused on "push-based" techniques that tradeoff result accuracy for reduced communication cost, by apportioning the error in the query answer across *filter* conditions installed locally at the sites to reduce communication.

Approximation and *randomization* techniques are also essential components of the distributed stream querying model, and play a critical role in minimizing communication. Approximate answers are often sufficient when tracking the statistical properties of large-scale distributed systems, since the focus is typically on indicators or patterns rather than precisely-defined events. This is a key observation, allowing for techniques that effectively tradeoff efficiency and approximation accuracy.

(2) *Communication Model:* The architecture and characteristics of the underlying communication network have an obvious impact on the design of effective distributed stream processing techniques. Most existing work has focused on *hierarchical* (i.e., tree) network architectures, due to both their conceptual simplicity and their importance for practical scenarios (e.g., sensornet routing trees [18]). As an example, Figure 1(a) depicts a simple *single-level* hierarchical model with $m + 1$ sites and n (distributed) update streams. Stream updates arrive continuously at the remote sites $1, \ldots, m$, whereas site 0 is a special *coordinator* site that is responsible for generating answers to (one-shot or 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.

More general, *multi-level* hierarchies have individual substream-monitoring sites at the leaves and internal nodes of a general *communication tree*, and the goal is to effectively answer or track a stream query $Q(S_1, \ldots, S_n)$ at the *root node* of the tree. The most general setting are *fully-distributed* models, 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 answer of a distributed stream query.

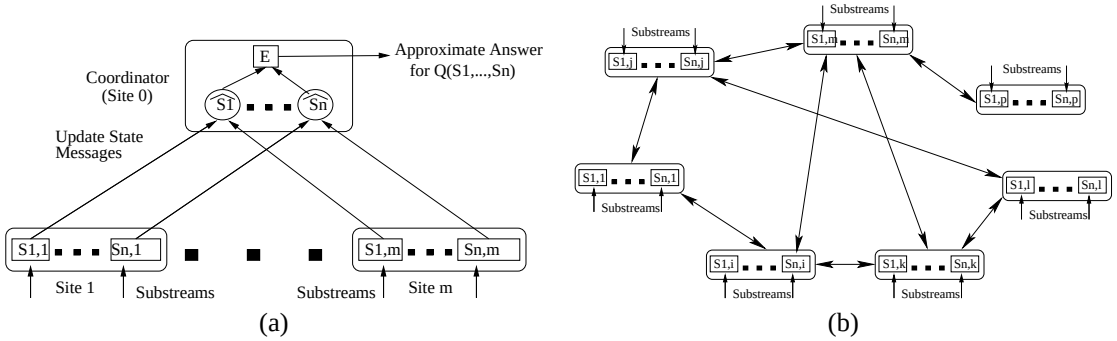

Figure 1: (a) Single-level hierarchical stream-processing model. (b) Fully-distributed model.

Besides the connectivity model, other important network characteristics for distributed stream processing include: the potential for broadcasting or multicasting messages to sites (e.g., over a limited radio range as in wireless sensornets), and the node/link-failure and data-loss characteristics of the supporting hardware.

(3) *Class of Queries:* The key dichotomy between simple, *non-holistic* aggregate queries (e.g., `MIN`, `SUM`, `AVG`) and *holistic* aggregates (e.g., median) has already been discussed; clearly, holistic aggregates introduce many more challenges for efficient distributed streaming computation. Another important distinction is that between *duplicate-sensitive* aggregates (that support bag/multi-set semantics, such as median, `SUM`, or top-k) and *duplicate-insensitive* aggregates (that support set semantics, such as `MIN` or count-distinct). Finally, another important class is that of complex *correlation* queries that combine/correlate streaming

data across different remote sites (e.g., through a streaming *join* computation). Such correlations can be critical in understanding important trends and making informed decisions about measurement or utilization patterns. Different classes of streaming queries typically require different algorithmic machinery for efficient distributed computation.

The remainder of this section provides a brief overview of some key results in distributed data streaming, for both the one-shot and continuous querying models, and concludes with a short survey of systems-related efforts in the area.

One-Shot Distributed Stream Processing. Madden et al. [18] present simple, exact *tree-based aggregation* schemes for sensor networks and propose a general framework based on *generate*, *fuse*, and *evaluate* functions for combining partial results up the aggregation tree. They also propose a classification of different aggregate queries based on different properties, such as duplicate in/sensitivity, example or summary results, monotonicity, and whether the aggregate is *algebraic* or *holistic* (which essentially translates to whether the intermediate partial state is of constant size or growing). While the exact computation of holistic aggregates requires linear communication cost, guaranteed-quality *approximate* results can be obtained at much lower cost by approximating intermediate results through *composable data synopses* [1, 9].

Robustness is a key concern with such hierarchical aggregation schemes, as a single failure/loss near the root of the tree can have a dramatic effect on result accuracy. *Multi-path routing* schemes mitigate this problem by propagating partial results along multiple different paths. This obviously improves reliability and reduces the impact of potential failures; in addition, this improved reliability often comes essentially "for free" (e.g., in wireless sensornets where the network is a natural broadcast medium). Of course, multi-path routing also implies that the same partial results can be accounted for multiple times in the final aggregate. As observed by Nath et al. [20], this duplication has no effect on aggregates that are naturally *Order and Duplicate Insensitive (ODI)*, such as `MIN` and `MAX`; on the other hand, for non-ODI aggregates, such as `SUM` and `COUNT`, *duplicate-insensitive sketch synopses* (e.g., based on the Flajolet-Martin sketch [9]) can be employed to give effective, low-cost, multi-path approximations [20]. Hybrid approaches combining the simplicity of tree aggregation (away from the root node) and the robustness of multi-path routing (closer to the root) have also been explored [19].

Gossip (or, *epidemic*) protocols for spreading information offer an alternative approach for robust distributed computation in the more general, fully-distributed communication model (Figure 1(b)). ODI aggregates (and sketches) naturally fit into the gossiping model, which basically guarantees that all n nodes of a network will converge to the correct global ODI aggregate/sketch after $O(\log n)$ rounds of communication. For non-ODI aggregates/sketches, Kempe et al. [15] propose a novel gossip protocol (termed *push-sum*) that also guarantees convergence in a logarithmic number of rounds, and avoids double counting by splitting up the aggregate/sketch and ensuring "conservation of mass" in each round of communication.

Continuous Distributed Stream Processing. The continuous model places a much more stringent demand on the distributed stream processing engine, since remote sites must collaborate to *continuously* maintain a query answer that is accurate (e.g., within specified error bounds) based on the current state of the stream(s). Approximation plays a critical role in the design of communication-efficient solutions for such *continuous monitoring* tasks. In a nutshell, the key idea is to tradeoff result accuracy and local processing at sites for reduced communication costs, by installing *local filters* at the remote sites to allow them to only "push" significant updates to the coordinator; of course, these distributed local filters would have to be *safe*, that is, they should guarantee the overall error bound for the global query result (based on the exact current state) at the coordinator. This idea of local traffic filtering for continuous distributed queries is pictorially depicted in Figure 2(a).

A key concept underlying most continuous distributed monitoring schemes is that of *adaptive slack allocation* — that is, adaptively distributing the overall "slack" (or, error tolerance) in the query result across the local filters at different participating sites based on observed local update patterns. Obviously, the complexity of such slack-distribution mechanisms depends on the nature of the aggregate query being tracked. Olston et al. [21] consider the simpler case of algebraic aggregates (where breaking down the overall slack to safe local filters is straightforward), and discuss adaptive schemes that continuously grow/shrink local filters based on the frequency of observed local violations. As expected, the situation is more complicated in the case of holistic aggregates: Babcock and Olston [2] discuss a scheme for tracking an approximate global top-k set of items using a cleverly-built set of local constraints that essentially "align" the *local* top-k set at a site with the *global* top-k; furthermore, their algorithm also retains some amount of slack at the coordinator to allow for possible localized resolutions of constraint violations. Das et al. [8] consider the problem of monitoring distributed set-expression cardinalities and propose tracking algorithms that take advantage of the set-expression semantics to appropriately "charge" updates arriving at the local sites.

Simple slack-allocation schemes are typically based on a naive *static model* of local-site behavior; that is, the site's "value" is assumed constant since the last update to the coordinator, and communication is avoided as long as this last update value stays within the slack bounds. Cormode and Garofalakis [5] propose the use of more sophisticated, *dynamic prediction models* of temporal site dynamics in conjunction with appropriate sketching techniques for communication-efficient monitoring of complex distributed aggregate queries. Their idea is to allow each site and the coordinator to share a prediction of how the site's local stream(s) (and, their sketch synopses) evolve over time. The coordinator uses this prediction to provide continuous query answers, while the remote site checks locally that the prediction stays "close" to the actual observed streaming distribution

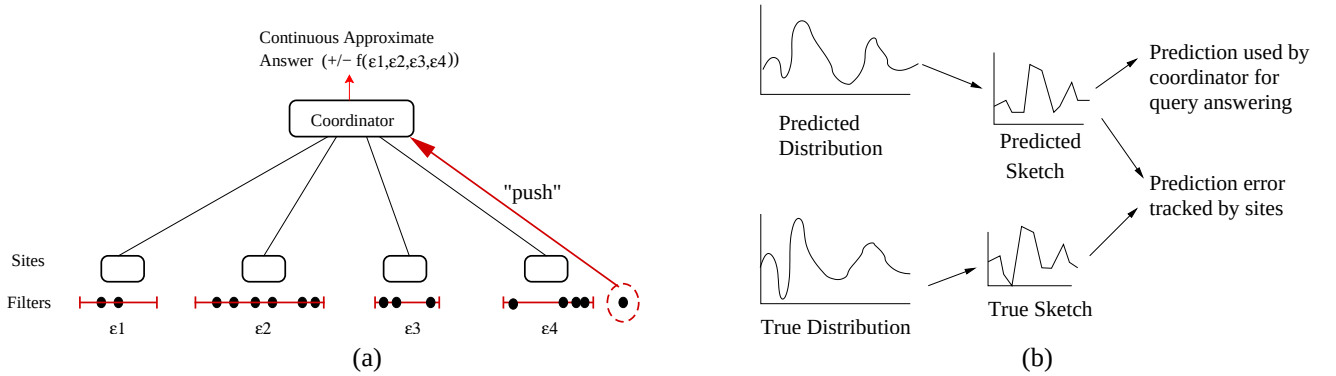

Figure 2: (a) Using local filters for continuous distributed query processing: Most updates fall within the local-filter ranges and require no communication with the coordinator (that can provide approximate answers with guarantees depending on the filter "widths"); only updates outside the local-filter range require new information to be "pushed" by the local site to the coordinator. (b) Prediction-based approximate query tracking: Predicted sketches are based on simple prediction models of local-stream behavior, and are kept in-sync between the coordinator (for query answering) and the remote sites (for tracking prediction error).

(Figure 2(b)). Of course, using a more sophisticated prediction model can also impose some additional communication to ensure that the coordinator's view is kept in-sync with the up-to-date local stream models (at the remote sites). Combined with intelligent sketching techniques and methods for bounding the overall query error, such approaches can be used to track a large class of complex, holistic queries, only requiring concise communication exchanges when prediction models are no longer accurate [5]. Furthermore, their approach can also be naturally extended to multi-level hierarchical architectures. Similar ideas are also discussed by Chu et al. [4] who consider the problem of *in-network probabilistic model maintenance* to enable communication-efficient approximate tracking of sensornet readings.

A common feature of several distributed continuous monitoring problems is continuously evaluating a condition over distributed streaming data, and *firing* when the condition is met. When tracking such *distributed triggers*, only values of the "global" continuous query that are above a certain threshold are of interest (e.g., fire when the total number of connections to an IP destination address exceeds some value) [13]. Recent work has addressed versions of this distributed triggering problem for varying levels of complexity of the global query, ranging from simple counts [16] to complex functions [26] and matrix-analysis operators [12]. Push-based processing using local-filter conditions continues to play a key role for distributed triggers as well; another basic idea here is to exploit the threshold to allow for even more effective local traffic filtering (e.g., "wider" yet safe filter ranges when the query value is well below the threshold).

Systems and Prototypes. Simple, algebraic in-network aggregation techniques have found widespread acceptance in the implementation of efficient sensornet monitoring systems (e.g., TAG/TinyDB [18]). On the other hand, more sophisticated approximate in-network processing tools have yet to gain wide adoption in system implementations. Of course, Distributed Stream-Processing Engines (DSPEs) are still a nascent area for systems research: only a few research prototypes are currently in existence (e.g., Telegraph/TelegraphCQ [25], Borealis/Medusa [3], P2 [17]). The primary focus in these early efforts has been on providing effective system support for *long-running stream-processing dataflows* (comprising connected, pipelined query operators) over a distributed architecture (Figure 3). For instance, Balazinska et al. [3] and Shah et al. [25] discuss mechanisms and tools for supporting parallel, highly-available, fault-tolerant dataflows; Loo et al [17] propose tools for declarative dataflow design and automated optimizations; Pietzuch et al. [22] consider the problem of distributed dataflow operator placement and propose techniques based on a cost-space representation that optimize for network-efficiency metrics (e.g., bandwidth, latency); finally, Xing et al. [27] give tools for deriving distributed dataflow schedules that are resilient to load variations in the input data streams. To deal with high stream rates and potential system overload, these early DSPEs typically employ some form of *load shedding* [3] where tuples from operators' input stream(s) are dropped (either randomly or based on different QoS metrics). Unfortunately, such load-shedding schemes cannot offer any hard guarantees on the quality of the resulting query

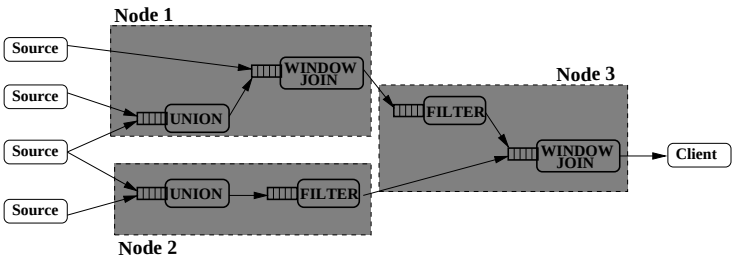

Figure 3: Distributed stream-processing dataflow.

answers. A mechanism based on *revision tuples* can be employed in the Borealis DSPE to ensure that results are *eventually correct* [3]. AT&T's Gigascope streaming DB for large-scale IP-network monitoring [7] uses approximation tools (e.g., sampling, sketches) to efficiently track "line-speed" data streams at the monitoring endpoints, but has yet to explore issues related to the physical distribution of the streams and holistic queries.

KEY APPLICATIONS

Enterprise and ISP Network Security: The ability to efficiently track network-wide traffic patterns plays a key role in detecting anomalies and possible malicious attacks on the network infrastructure. Given the sheer volume of measurement data, continuously centralizing all network statistics is simply not a feasible option, and distributed streaming techniques are needed.

Sensornet Monitoring and Data Collection: Tools for efficiently tracking global queries or collecting all measurements from a sensornet have to employ clever in-network processing techniques to maximize the lifetime of the sensors.

Clickstream and Weblog Monitoring: Monitoring the continuous, massive streams of weblog data collected over distributed web-server collections is critical to the real-time detection of potential system abuse, fraud, and so on.

FUTURE DIRECTIONS

The key algorithmic idea underlying the more sophisticated distributed data-stream processing techniques discussed in this article is that of effectively trading off space/time *and communication* with the quality of an approximate query answer. Exploring some of the more sophisticated algorithmic tools discussed here in the context of real-life systems and applications is one important direction for future work on distributed streams; other challenging areas for future research, include:

- Extensions to other application areas and more complex communication models, e.g., monitoring P2P services over shared infrastructure (OpenDHT [23] over PlanetLab), and dealing with constrained communication models (e.g., intermittent-connectivity and delay-tolerant networks (DTNs) [14]).
- Richer classes of distributed queries, e.g., set-valued query answers, machine-learning inference models [11].
- Developing a theoretical/algorithmic foundation of distributed data-streaming models: What are fundamental lower bounds, how to apply/extend information theory, communication complexity, and distributed coding. Some initial results appear in the recent work of Cormode et al. [6].
- Richer prediction models for stream tracking: Can models effectively capture site correlations rather than just local site behavior? More generally, understand the model complexity/expressiveness tradeoff, and come up with principled techniques for capturing it in practice (e.g., using the MDL principle [24]).
- Stream computations over an *untrusted* distributed infrastructure: Coping with privacy and authentication issues in a communication/computation-efficient manner. Some initial results appear in [10].

DATA SETS

Publicly-accessible network-measurement data collections can be found at the Internet Traffic Archive: (`http://ita.ee.lbl.gov/`), and CRAWDAD (the Community Resource for Archiving Wireless Data at Dartmouth, `http://cmc.cs.dartmouth.edu/data/dartmouth.html`).

CROSS REFERENCES

DATA STREAMS, STREAM PROCESSING, STREAM MODELS, CONTINUOUS QUERY, SYNOPSIS STRUCTURES, AMS SKETCH, COUNT-MIN SKETCH, STREAM SAMPLING, LOAD SHEDDING, STREAMING APPLICATIONS, SCHEDULING STRATEGIES

RECOMMENDED READING

[1] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, Philadelphia, Pennsylvania, May 1996.

[2] Brian Babcock and Chris Olston. "Distributed Top-K Monitoring". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2003.

[3] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Michael Stonebraker. "Fault-Tolerance in the Borealis Distributed Stream Processing System". In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, June 2005.

[4] David Chu, Amol Deshpande, Joseph M. Hellerstein, and Wei Hong. "Approximate Data Collection in Sensor Networks using Probabilistic Models". In *Proceedings of the 22nd International Conference on Data Engineering*, Atlanta, Georgia, April 2006.

[5] Graham Cormode and Minos Garofalakis. "Sketching Streams Through the Net: Distributed Approximate Query Tracking". In *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, September 2005.

[6] Graham Cormode, S. Muthukrishnan, and Ke Yi. "Algorithms for Distributed Functional Monitoring". In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, January 2008.

[7] Chuck Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. "Gigascope: A Stream Database for Network Applications". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2003.

[8] Abhinandan Das, Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. "Distributed Set-Expression Cardinality Estimation". In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, September 2004.

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

[10] Minos Garofalakis, Joseph M. Hellerstein, and Petros Maniatis. "Proof Sketches: Verifiable In-Network Aggregation". In *Proceedings of the 23rd International Conference on Data Engineering*, Istanbul, Turkey, April 2006.

[11] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. "Distributed Regression: An Efficient Framework for Modeling Sensor Network Data". In *Information Processing in Sensor Networks*, 2004.

[12] Ling Huang, XuanLong Nguyen, Minos Garofalakis, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, and Nina Taft. "Communication-Efficient Online Detection of Network-Wide Anomalies". In *Proceedings of IEEE INFOCOM'2007*, Anchorage, Alaska, May 2007.

[13] 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.

[14] Sushant Jain, Kevin Fall, and Rabin Patra. "Routing in a Delay Tolerant Network". In *Proceedings of ACM SIGCOMM*, Philadelphia, Pennsylvania, August 2005.

[15] David Kempe, Alin Dobra, and Johannes Gehrke. "Gossip-Based Computation of Aggregate Information". In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, October 2003.

[16] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. "Communication-efficient distributed monitoring of thresholded counts". In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 289–300, Chicago, Illinois, June 2006.

[17] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. "Declarative Networking: Language, Execution, and Optimization". In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, June 2006.

[18] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks". In *Operating Systems Design and Implementation (OSDI)*, Boston, Massachusetts, December 2002.

[19] A. Manjhi, S. Nath, and P. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2005.

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

[21] Chris Olston, Jing Jiang, and Jennifer Widom. "Adaptive Filters for Continuous Queries over Distributed Data Streams". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2003.

[22] Peter Pietzuch, Jonathan Ledlie, Jeffrey Schneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. "Network-Aware Operator Placement for Stream-Processing Systems". In *Proceedings of the 22nd International Conference on Data Engineering*, Atlanta, Georgia, April 2006.

[23] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu Yu. "OpenDHT: A Public DHT Service and its Uses". In *Proceedings of ACM SIGCOMM*, Philadelphia, Pennsylvania, August 2005.

[24] J. Rissanen. "Modeling by shortest data description". *Automatica*, 14:465–471, 1978.

[25] Mehul A. Shah, Joseph M. Hellerstein, and Eric Brewer. "Highly Available, Fault-Tolerant, Parallel Dataflows". In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, France, June 2004.

[26] Izchak Sharfman, Assaf Schuster, and Daniel Keren. "A geometric approach to monitoring threshold functions over distributed data streams". In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 301–312, Chicago, Illinois, June 2006.

[27] Ying Xing, Jeong-Hyon Hwang, Ugur Cetintemel, and Stan Zdonik. "Providing Resiliency to Load Variations in Ditributed Stream Processing". In *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul, Korea, September 2006.