Group-aware Stream Filtering

Ming Li and David Kotz
Department of Computer Science, Dartmouth College
Hanover, NH 03755
{ mingli, dfk} at cs.dartmouth.edu

Abstract

In this paper we are concerned with disseminating high-volume data streams to many simultaneous context-aware applications over a low-bandwidth wireless mesh network. For bandwidth efficiency, we propose a group-aware stream filtering approach, used in conjunction with multicasting, that exploits two overlooked, yet important, properties of these applications: 1) many applications can tolerate some degree of "slack" in their data quality requirements, and 2) there may exist multiple subsets of the source data satisfying the quality needs of an application. We can thus choose the "best alternative" subset for each application to maximize the data overlap within the group to best benefit from multicasting. An evaluation of our prototype implementation shows that group-aware data filtering can save bandwidth with low CPU overhead.

keywords: data dissemination, overlay multicasting, data filtering, bandwidth reduction

1. Introduction

Distributed context-aware applications may need to subscribe to remote information sources that provide contextual information about the users' locations and their environment. In some applications, context data collected by sensor networks often come to the subscribers as high-volume data streams. Transporting context "streams" via wired links may not be feasible in hard-to-wire areas, such as on a disaster scene; a wireless mesh network is a more cost-effective alternative. According to many studies, however, the effective bandwidth in a wireless mesh network is usually much lower than its wired counterpart, due to congestion and the limits of radio communications. Hence, there is a disparity between the high data demands of context dissemination in a wireless network and its limited bandwidth.

Two main approaches have been proposed to tackle the problem. One is to eliminate redundant communications with multicast protocols when disseminating common data to multiple subscribers. The other is to reduce the data at a context source, by applying application-specific filters at the source node to select only those tuples "important" for meeting the applications' data-quality requirements. Since source-sharing applications may use context source in different ways, the filters deployed at the same source may select different portions of the source data. If there is sufficient overlap of the data selected by the filters, we can still multicast the data to further reduce bandwidth demands. Thus, at the source node, we multiplex the filtered streams to form a multicast stream. Figure 1 shows this process: two applications, A and B, share the same context source $< D1, D2, D3, ... >$, but each application's filter selects a different subset. The multicast protocol allows us to label each tuple with the list of the applications that should receive that tuple; thus each tuple is transmitted at most once on any link.

We here propose a solution that combines multicasting and filtering for context streams. In contrast to self-interested filtering, which only considers each individual application's needs, we propose *group-aware stream filtering* that considers the needs of individual applications, as well as those of other subscribers. The result of this "group-aware stream filtering" satisfies all subscribers' data requirements, and simultaneously ensures maximum data sharing among the subscribers to make the best use of a multicast protocol in saving bandwidth. Our work makes use of two overlooked, yet important, properties of context-aware applications: 1) many applications can tolerate some degree of "slack" in their data quality requirements, and 2) there may exist multiple subsets of the source data satisfying the quality needs of an application. We can thus choose the "best alternative" subset for each application to maximize the data overlap within the group to best benefit from multicasting.

In the paper, we describe the following contributions of this work.

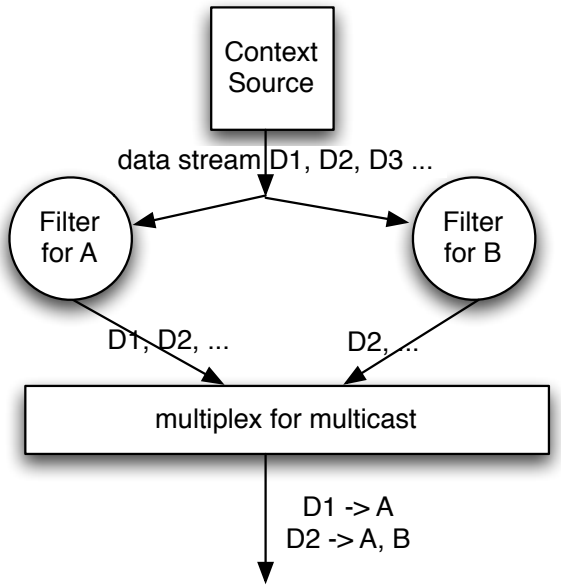

Figure 1. Multiplexing filtered streams for multicasting.

- Our approach uses multicast protocols in concert with data filtering to reduce unnecessary data traffic, which is crucial for a wireless dissemination system to support large-scale context sharing. The core of our approach is to exploit semantics of applications to reduce data communication. We treat saving bandwidth a goal as important as providing data to satisfy applications' quality needs.
- We developed a framework that encapsulates the general idea of group-aware filtering.
- We built a prototype system for evaluation. Our preliminary experiments show that this approach can effectively save bandwidth with low CPU overhead.

In the rest of the paper, we first describe in Section 2 the basis for group filtering, and introduce our framework for the group-aware stream filtering in Section 3. In Section 4, we discuss our evaluation of this concept based on a prototype implementation. We discuss related work in Section 5 and conclude in Section 6 with a description of future work.

2. Two key observations

In this section, we make two key observations about stream filtering for context-aware applications. The observations motivate our "group-aware stream filtering" approach detailed in the next section.

2.1. Quality requirements of stream filtering

The goal of stream filtering is to select an "important" portion from a streaming data source according to the specific needs of an application. The result of this filtering reflects an applications' desirable data quality, which is normally measured as the accuracy, granularity, timeliness, or completeness of the data. For example, an application would like to get a temperature reading of a place whenever the reading has changed by n degrees. This n-degree data granularity requirement can be enforced by a "delta-compression" filter that keeps the temperature state and compresses the streaming data at "delta", in this case n unit, intervals.

2.2. First observation

The first key observation we have made about the context-aware applications is that they may tolerate some degree of "slack" in their data quality.

Consider a temperature source and delta-compression filtering for example. Given a time-ordered nine-tuple sequence from the source: $< \quad 0, 35, 29, 45, 50, 59, 80, 97, 100 \quad >$, the output that satisfies compression at 50-unit granularity (here we assume the initial state is the first reading of the temperature) will be $< \quad 0, 50, 100 \quad >$.[1] We recognize that applications may find it harmless to tolerate a small deviation from the ideal compression granularity in the output. For instance, the application may be able to tolerate a maximum of 10-degree "slack" with regard to its ideal 50-degree granularity requirement. The quality deviation can be specified in various ways: one may specify the tolerable degree of deviation in reference to data points in the source stream that perfectly satisfy the targeted data granularity, or one may specify the slack using distance functions or membership functions of some state important to the application.

2.3. Second observation

Our second observation is that more than one sequence from a data source can potentially satisfy an application's approximate quality requirements.

In the previous example, if the application tolerates a maximum of 10-degree slack in the 50-degree com-

[1] Here we represent each tuple as a single integer; in reality, each tuple may have several fields, but for simplicity we represent each by the value of its "temperature" field since it is that field that is used for filtering.

pression granularity, it is easy to validate that the following sequences satisfy the approximate granularity requirements as well: $< 0, 45, 100 >$, $< 0, 59, 100 >$, $< 0, 50, 97 >$, $< 0, 45, 97 >$, $< 0, 59, 97 >$.

2.4. Group-awareness

Let us call the above delta-compression application A. Suppose application B shares the same source as A and tolerates a maximum of 5-degree slack in the 40-degree compression granularity. By the same token, it is also easy to validate that the following sequences satisfy B's requirements: $< 0, 45, 97 >$, $< 0, 50, 97 >$, $< 0, 50, 100 >$, $< 0, 45, 100 >$.

Individually, A may choose $< 0, 50, 100 >$ as its output; B may choose $< 0, 45, 97 >$ as its output. This makes a total of 5 tuples to output when multiplexing the output streams for multicasting. If A and B are aware of each other's filtering needs, and both decide on, say $< 0, 50, 97 >$, as their individual output, then only three tuples need to be multicast to A and B to satisfy both filtering requirements. In effect, the "group-awareness" reduces the bandwidth demand by two tuples.

3. Framework for group-aware stream filtering

In this section, we formally define the problem the group-aware stream filtering tries to solve and show a general group-aware filtering algorithm whose basic idea we have briefly shown in the previous example. First, we will introduce the assumptions and background of our work.

3.1. Definition of the problem

With the above background information, we now formally define the group-aware stream filtering problem as the following optimization problem.

Consider a source stream S and n filters $Op_1, Op_2, ...Op_n$ deployed at the source node for n subscribing applications Application 1, Application 2, ... Application n, respectively. Filters process source data in time-progressive batches or *data windows*. An output of a filter is a time-ordered sequence. For simplicity, we assume the timestamp of each tuple in the sequence is unique and thus the output can be represented as a set of tuples. We define the set that contains all satisfying outputs of Application i based on a data window $S_w \subset S$ as $PotentialOut_i = \{S' | S' \subset S_w, S'$ satisfy application i's quality requirements$\}$. The goal of grouped filtering is for each Op_i to pick an element out_i from $PotentialOut_i (i = 1..n)$ such that $|out_1 \bigcup out_2 ... \bigcup out_n|$ is minimized (see Figure 2).

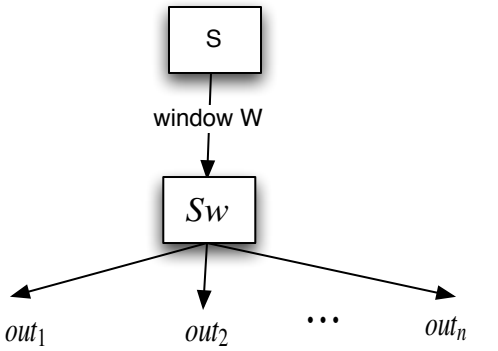

Figure 2. Group-aware stream filtering problem

3.2. Framework for group-aware stream filtering

Saving bandwidth is important to satisfy long-running applications' quality needs and for the system to scale well to a large number of co-existing applications that may cooperate for a common mission. Thus, applications deployed in a wireless bandwidth-conscious network are motivated to expose their approximate data needs for the system to reduce the overall bandwidth demand. Each application may reveal "quality-equivalent" candidates for each of its outputs based on a "reference point-based" approach. We define *reference points* as the output that a self-interested filter would normally produce. Then, applications can define a "slack" of a reference point to include all adjacent data points that are "slack" units away from the reference point as its candidate set. For instance, in the 50-degree delta-compression example mentioned in Section 2, the reference points of the 9-tuple sequence are $0, 50, 100$. If the application has a 10-degree "slack", we can identify the candidate replacement for each reference points by computing the contiguous range of tuples before or after the desired reference points, as long as each value is no more than 10 degree below or over that of the reference point. For instance, 50 now has a candidate set consisting of $45, 50$.

In our framework, applications can declare quality "slack" with distance or membership functions. In the delta-compression example, a numeric temperature difference defines the slack. A distance function may use other attributes as well, such as the timestamp of the tuples. It may also involve multiple attributes of the data. In a location trace, for example, the distance function

```
GROUPAWARE-STREAM-FILTER(S, qSpec)
    ▷ initialize the filter's internal state with application's quality specification qSpec
1   internalState ← qSpec.initInternalState();
2   while (( currentTuple ← S.getNextTuple())! = null);
3       do if isAdmissable(currentTuple, qSpec, internalState)
4           then
                ▷ first stage: get candidates
5               internalState ← addToCandidateSet(currentTuple);
6               internalState.update(currentTuple)
7               globalState.update("groupUtil", currentTuple);
8           else if closeCandidateSet(currentTuple, qSpec, internalState)
9               then
                    ▷ second stage: decide output for this application
10                  output ← decideOutput(globalState, internalState);
                    ▷ record the output in global state
11                  globalState.update("dataForMulticast",output);
```

Figure 3. Group-aware stream filtering algorithm

may be the Euclidean function involving two or three attributes that describe a location. If a context stream represents observations made by many sensing devices, an application can declare a membership function for the sensing devices based on the similarity in their sensing capacity and environment, such that the observations made by a member sensor can be treated equivalent to those made by any other member.

We abstract the group-aware filtering process into the following continuous two-stage process at each filter.

1. **First stage: finding candidates for a reference point**: select a candidate set that contains tuples that can potentially satisfy the data quality requirements of the application. Communicate the candidate set to other source-sharing applications via global state.

2. **Second stage: deciding the output**: With reference to the global state, pick a subset of tuples from the candidate set for output. Communicate the choices via global state.

Finally, merge and multicast the chosen output tuples to the subscribing applications. The global state in our framework consists of 1) the *group utility* of each tuple, which captures how many applications have the tuple in their candidate set, and 2) *data-for-multicast* which records each application's already-decided outputs that have not yet been multicast. This two-stage process is shown in GROUPAWARE-STREAM-FILTER of Figure 3.

This process takes the source stream S and application's quality specification $qSpec$ as inputs. The $qSpec$ includes the predicates and functions that can be invoked by procedure $isAdmissable$ and $closeCandidateSet$ to build the candidate set for a reference output. $qSpec$ also initializes the internal state of the filter. The "groupUtil" field of the global state is incremented when a tuple can be included in the candidate set of an application; the "dataForMulticast" field of the global state object is to record the chosen output tuples, which will be multicast later. The rules for choosing tuples to output from a candidate set are captured in $decideOutput$.

The software architecture of the framework shown in Figure 4 consists of the following modules: 1) the application specification manager, which facilitates applications to specify their data needs with a library of common predicates, distance functions, etc.; 2) the group-aware stream filtering manager, which manages a pool of group-aware filters instantiated according to the applications' quality specifications; 3) an output scheduler that merges output decided by each application into the multicasting format before invoking the overlay multicasting protocol; 4) a global state manager that maintains the state information shared by the applications.

4. Evaluation

We evaluate group-aware stream filtering with a prototype implementation. We integrate our prototype with a general data dissemination system, *Solar* [5], developed at Dartmouth College. Solar is a distributed software infrastructures that can be deployed over a wireless mesh network to assist applications to collect, aggregate, and disseminate contextual data. Filters, as part of the

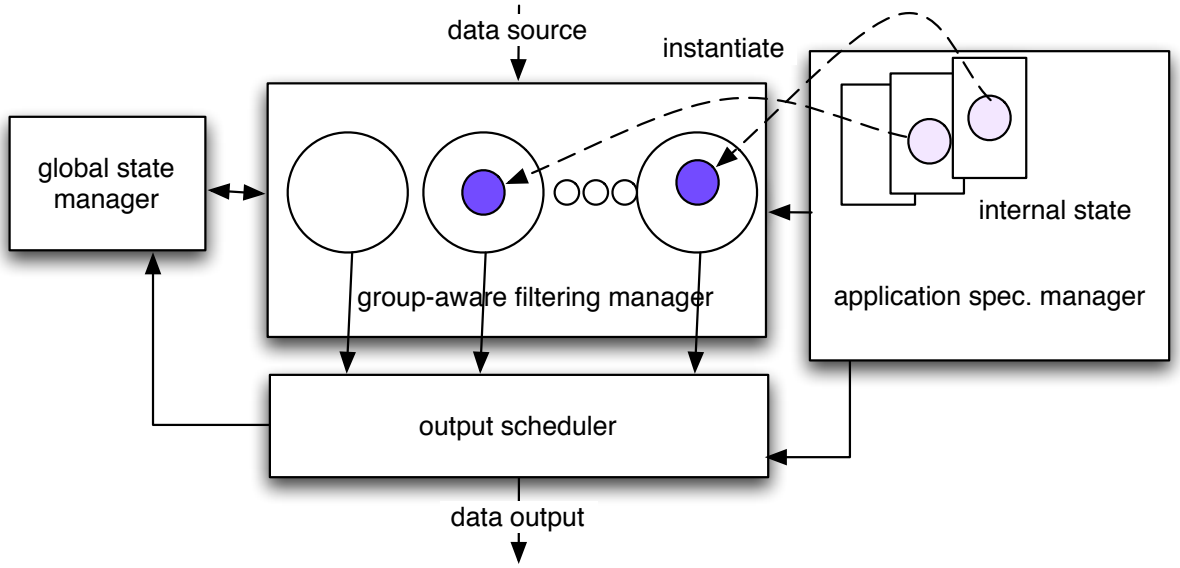

Figure 4. Framework for group-aware stream filtering

aggregation-oriented services, are deployed for applications on the nodes where context sources are published. The core of Solar is a scalable and self-organizing peer-to-peer overlay to support key data-driven services, including the application-level multicasting service. The multicast service uses its peer-to-peer routing substrate to improve the scalability of data dissemination and does not rely on IP multicast. Our group-aware filtering service utilizes both filtering and the multicast service provided by Solar. We have implemented group-aware stream filtering as an enhanced service running on top of Solar's basic filtering and multicasting services.

We use real sensor data collected by the NAMOS (Networked Aquatic Microbial System) project at UCLA[2] for data sources. In particular, we use the temperature trace collected in August 2006 as the major source of data for testing our ideas. That trace contains temperature samples every 10ms; it is important to save bandwidth while disseminating such data over a bandwidth-limited wireless mesh network. The temperature change of consecutive readings in the trace is, on average, well below 0.03 degree and it is reasonable to compress the data using the delta-compression specifications shown in Table 1.

We used Emulab[3] to set up a seven-node DHT ring for Solar overlay services. The nodes are 600MHz CPUs with FreeBSD 4.5 and JDK 1.4.2. We set the link capacity of the network to be 50Mbps (note that 50Mbps is much larger than effective bandwidth usually obtainable in a wireless mesh network. We set it so, mainly to get a lower-bound estimation on the transport latency among the nodes.) We created a Solar source that continuously publishes events obeying the original event interval in the temperature trace. We deployed two group-aware filters on the node that runs the source. The filters capture applications' delta-compression specifications shown in Table 1.

[2]http://cens.ucla.edu
[3]http://www.emulab.net

Table 1. Delta-compression specifications

Spec.	App1(Delta/Slack)	App2(Delta/Slack)
spec1	0.03/0.003	0.04/0.004
spec2	0.03/0.006	0.04/0.008
spec3	0.03/0.012	0.04/0.016
spec4	0.03/0.006	0.06/0.012

We evaluate the bandwidth savings of our approach using the *output ratio*, which is the total number of tuples output by the group-aware filters over the total number of tuples output by individual filters. This metric measures the bandwidth saved by group filtering beyond that saved by multicasting and basic filtering. Figure 5 shows the results; for each test, we show the average and the median of the output ratio, across batches of 100 tuples. We can see that group-aware filtering can reduce output as much as 20% in these tests. For a low-bandwidth network, such savings of bandwidth is valuable.

Intuitively, the wider the slack is in the delta-compression specification, the more likely it is to find

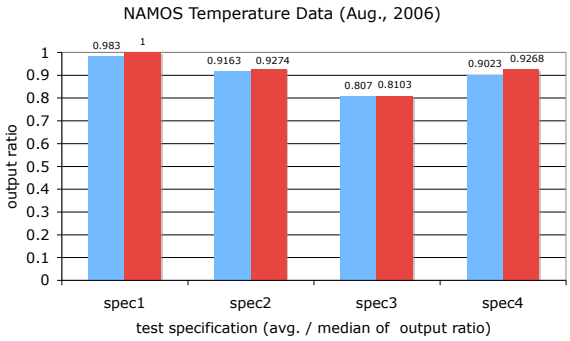

Figure 5. Average and median of output ratio

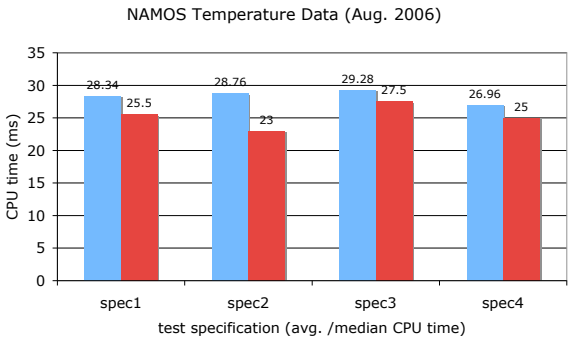

Figure 6. CPU cost in group-aware filtering

overlapped candidate sets for two applications, and thus output should be more reduced. This effect is reflected in our results: given the same delta values, $spec2$ has twice the slack of $spec1$, and $spec3$ has twice the slack of $spec2$, which explains the better output ratio of $spec2$ (0.9163) over $spec1$ (0.9830), and $spec3$ (0.8070) over $spec2$ (0.9163).

We measured the cost of group-aware filtering by the CPU time consumed per batch of data. Figure 6 shows the CPU cost of running the group-aware filtering algorithm in four settings. We can see that processing a batch of 100 tuples costs no more than 30ms in average: the group-aware filter processing rate was about 0.3ms per tuple, which is much faster than the data arrival rate (10ms per tuple). This shows that the group-aware filtering was suitable for fast stream processing thanks to its low CPU overhead.

We also measured the average latency of transporting a tuple from a node to another in the Emulab setup. It was about 134.01ms, which, compared with the 0.3ms per tuple processing rate of the group-aware filtering, is much more substantial. In a wireless mesh network whose effective bandwidth is well below 50Mbps, the even longer latency incurred during transportation makes the overhead of group-aware filtering negligible.

5. Related work

Bandwidth-reduction mechanisms, such as sampling, summarising, and filtering, have been actively researched in recent years in the systems community [11, 8, 2, 3, 4]. Most of the mechanisms are discussed in the context of a single streaming application. Only a few research efforts have looked into group optimization for streaming applications, but these mechanisms are either based on traditional compiler rewriting techniques, or the simple grouping of stateless filters [1, 6, 10, 12, 7]. When data reduction is based on simple filters, grouping the filters has been proved to save computational power [10, 12]. Different from their objectives, the goal of our work is to trade CPU time for bandwidth savings.

Our work exploits semantics of a stream processing application to improve resource management in a dissemination system. IBM's Gryphon [13] also uses the semantics of aggregation functions of the application to compress a sequence of data into a reduced sequence that will have the same effect on application's states. Zhao et al. [14] proposed a case-based language to specify an application's sophisticated processing needs, which identify what sequences are semantically equivalent to applications so that the system can re-order sequences and compensate the data lost in the network. We have a different goal than either project: rather than using a complicated language to describe the needs, we opt for a simple framework with libraries of distance functions and member functions to let applications describe the approximate nature of their data requirements.

Johnson et al. [9] summarized a general structure for sampling operators. The structure also contains stages, as we proposed. If we see our group-aware filtering from a sampling point of view, our algorithm is a special kind of sampler in that it picks an output from a candidate set of outputs. But our process involves coordination across a group of applications, which never occurs in Johnson's single-application oriented sampling.

6. Conclusion and future work

In a wireless data dissemination network, bandwidth limitations are a critical problem well recognized by the research community. In this work we let the data-dissemination system exploit the semantics of many streaming applications, in particular the applications' approximate nature of data filtering and processing, to save network bandwidth. The key of our solution is to leverage overlay multicasting and in-network *group-aware filters*. Our preliminary evaluations based on real sensor data traces collected by UCLA's CENS project have shown encouraging results with low CPU cost.

As a first step, we evaluated our work using a simple delta-compression filtering scheme. For future work, we would like to consider the following extensions.

- **Diverse data traces for testing.** We plan to accumulate more real event streams to further the evaluation. With experience, we will gain insights into the properties of applications and data that might allow us to improve group-aware filtering.
- **Diverse application needs for testing.** This work focused on the simple delta-compression needs of applications. We plan to test with more diverse filtering types. For example, applications may want to filter the data based on values of multiple attributes in a tuple or some patterns in the tuple sequence.
- **Topology-aware grouping.** Currently we group all the filters subscribing to a source, regardless of the topology of the applications. In scenarios where two applications share little of their routes from the source, there is little benefit to maximize the overlap of the two output streams. Hence, if we can group the applications' filters by how applications are clustered in the network, we may reduce the overall bandwidth needs even more.

7. Acknowledgments

The authors want to thank Guanling Chen, Kazuhiro Minami, other members in the ARTEMIS project at the Institutue for Security Technology Studies and other members of Center for Mobile Computing at Dartmouth College, for their valuable suggestions and feedback. This research program is a part of the Institute for Security Technology Studies, supported under Award number 2000-DT-CX-K001 from the U.S. Department of Homeland Security, Science and Technology Directorate, and by Grant number 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security or the United States Department of Justice.

References

[1] Suresh Aryangat, Henrique Andrade, and Alan Sussman. Time and space optimization for processing groups of multi-dimensional scientific queries. In *ICS '04: Proceedings of the 18th Annual International Conference on Supercomputing*, pages 95–105, New York, NY, USA, 2004. ACM Press.

[2] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA '02: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[3] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *STOC '01: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 266–275, New York, NY, USA, 2001. ACM Press.

[4] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 263–274, New York, NY, USA, 1999. ACM Press.

[5] Guanling Chen, Ming Li, and David Kotz. Design and implementation of a large-scale context fusion network. In *MobiQuitous '04: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems*, pages 246–255. ACMPress, 2004.

[6] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 379–390, New York, NY, USA, 2000. ACM Press.

[7] Reynold Cheng, Ben Kao, Sunil Prabhakar, Alan Kwan, and Yicheng Tu. Adaptive stream filters for entity-based queries with non-value tolerance. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 37–48. VLDB, 2005.

[8] Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 1–12, New York, NY, USA, 2005. ACM Press.

[9] Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2005. ACM Press.

[10] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 49–60, New York, NY, USA, 2002. ACM Press.

[11] Don P. Mitchell. Consequences of stratified sampling in graphics. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 277–280, New York, NY, USA, 1996. ACM Press.

[12] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD '03: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, San Diego, California, June 2003.

[13] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *International Symposium on Software Reliability Engineering (ISSRE '98)*, 1998.

[14] Yuanyuan Zhao and Rob Strom. Exploitng event stream interpretation in publish-subscribe systems. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 219–228, New York, NY, USA, 2001. ACM Press.