Issues in Data Stream Management*

Lukasz Golab and M. Tamer Özsu
University of Waterloo, Canada
{lgolab, tozsu}@uwaterloo.ca

Abstract

Traditional databases store sets of relatively static records with no pre-defined notion of time, unless timestamp attributes are explicitly added. While this model adequately represents commercial catalogues or repositories of personal information, many current and emerging applications require support for on-line analysis of rapidly changing data streams. Limitations of traditional DBMSs in supporting streaming applications have been recognized, prompting research to augment existing technologies and build new systems to manage streaming data. The purpose of this paper is to review recent work in data stream management systems, with an emphasis on application requirements, data models, continuous query languages, and query evaluation.

1 Introduction

Traditional databases have been used in applications that require persistent data storage and complex querying. Usually, a database consists of a set of objects, with insertions, updates, and deletions occurring less frequently than queries. Queries are executed when posed and the answer reflects the current state of the database. However, the past few years have witnessed an emergence of applications that do not fit this data model and querying paradigm. Instead, information naturally occurs in the form of a sequence (stream) of data values; examples include sensor data [10], Internet traffic [36, 61], financial tickers [17, 72], on-line auctions [3], and transaction logs such as Web usage logs and telephone call records [21].

A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Likewise, queries over streams run continuously over a period of time and incrementally return new results as new data arrive. These are known as *long-running*, *continuous*, *standing*, and *persistent* queries [17, 48]. The unique characteristics of data streams and continuous queries dictate the following requirements of data stream management systems:

- The data model and query semantics must allow order-based and time-based operations (e.g. queries over a five-minute moving window).
- The inability to store a complete stream suggests the use of approximate summary structures, referred to in the literature as *synopses* [6] or *digests* [72]. As a result, queries over the summaries may not return exact answers.
- Streaming query plans may not use blocking operators that must consume the entire input before any results are produced.
- Due to performance and storage constraints, backtracking over a data stream is not feasible. On-line stream algorithms are restricted to making only one pass over the data.
- Applications that monitor streams in real-time must react quickly to unusual data values.
- Long-running queries may encounter changes in system conditions throughout their execution lifetimes (e.g. variable stream rates).
- Shared execution of many continuous queries is needed to ensure scalability.

Proposed data stream systems resemble the abstract architecture shown in Figure 1. An input monitor may regulate the input rates, perhaps by dropping packets. Data are typically stored in three partitions: temporary working storage (e.g. for window queries), summary storage for stream synopses, and static storage for meta-data (e.g. physical location of each source). Long-running queries are registered

*This research is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

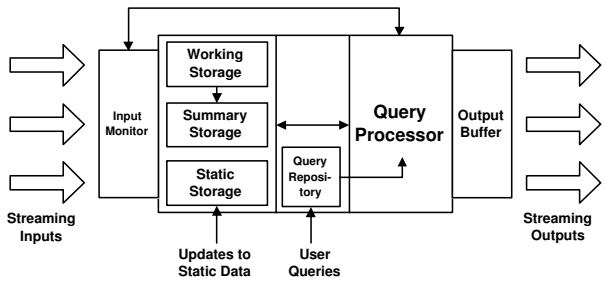

Figure 1: Abstract reference architecture for a data stream management system.

in the query repository and placed into groups for shared processing, though one-time queries over the current state of the stream may also be posed. The query processor communicates with the input monitor and may re-optimize the query plans in response to changing input rates. Results are streamed to the users or temporarily buffered.

In this paper, we review recent work in data stream processing, including data models, query languages, continuous query processing, and query optimization. Related surveys include Babcock et al. [6], which discusses stream processing issues in the context of the STREAM project, and a tutorial by Garofalakis et al. [31], which reviews algorithms for data streams. An extended version of this survey [39] includes more details.

The remainder of this paper surveys requirements of streaming applications (Section 2), models and query languages for data streams (Section 3), streaming operators (Section 4), and query processing and optimization (Section 5). We conclude in Section 6 with a list of academic projects related to data stream management.

2 Streaming Applications

We begin by reviewing a collection of data stream applications in order to define a set of query types that a data stream management system should support; more examples may be found in [60, 65].

2.1 Sensor Networks

Sensor networks may be used in various monitoring applications that involve complex filtering and activation of an alarm in response to unusual conditions. Aggregation and joins over multiple streams are required to analyze data from many sources, while aggregation over a single stream may be needed to compensate for individual sensor failures. Representative queries include the following:

- *Drawing temperature contours on a weather map:* Perform a join of temperature streams (on the temperature attribute) produced by weather monitoring stations. Join the results with a static table containing the latitude and longitude of each station, and connect all points that have reported the same temperature with lines.
- Analyze a stream of recent power usage statistics reported to a power station (group by location, e.g. city block), and adjust the power generation rate if necessary [18].

2.2 Network Traffic Analysis

Ad-hoc systems for analyzing Internet traffic in near-real time are already in use to compute traffic statistics and detect critical conditions (e.g. congestion and denial of service) [22, 36, 61]. Monitoring popular source and destination addresses is particularly important as Internet traffic patterns are believed to obey the Power Law distribution, meaning that most of the bandwidth is consumed by a small set of heavy users. Example queries include:

- *Traffic matrices*: Determine the total amount of bandwidth used by each source-destination pair, and group by protocol type or subnet mask.
- Compare the number of distinct source-destination pairs in the (logical) streams containing the second and third steps, respectively, of the three-way TCP handshake. If the counts differ by a large margin, then a denial-of-service attack may be taking place.

2.3 Financial Tickers

On-line analysis of stock prices involves discovering correlations, identifying trends and arbitrage opportunities, and forecasting future values [72]. The following are typical queries [63]:

- *High Volatility with Recent Volume Surge*: Find all stocks priced between $20 and $200, where the spread between the high tick and the low tick over the past 30 minutes is greater than 3% of the last price, and where in the last 5 minutes the average volume has surged by more than 300%.
- *NASDAQ Large Cap Gainers*: Find all NASDAQ stocks trading above their 200-day moving average with a market cap greater than $5 Billion that have gained in price today by at least 2%, and are within 2% of today's high.

2.4 Transaction Log Analysis

On-line mining of Web usage logs, telephone call records, and Automated Bank Machine transactions also conform to the data stream model. The goal is to find interesting customer behaviour patterns, identify suspicious spending behaviour that could indicate fraud, and forecast future data values. The following are some examples:

- Examine Web server logs in real-time and reroute users to backup servers if the primary servers are overloaded.
- *Roaming diameter* [21]: Mine cellular phone records and for each customer, determine the greatest number of distinct base stations used during one telephone call.

2.5 Analysis of Requirements

The preceding examples show significant similarities in data models and basic operations across applications (and some differences related to workload characteristics, such as stream arrival rates or the amount of historical data needed). We list below a set of fundamental continuous query operations over streaming data, keeping in mind that new streaming applications, possibly with additional requirements, will likely be proposed in the future.

- *Selection*: All streaming applications require support for complex filtering.
- *Nested aggregation*: Complex aggregates, including nested aggregates (e.g. comparing a minimum with a running average) are needed to compute trends in the data.
- *Multiplexing and demultiplexing*: These are similar to group-by and union, respectively, and are used to decompose and merge logical streams.
- *Frequent item queries*: These are also known as *top-k* or *threshold* queries, depending on the cut-off condition.
- *Stream mining*: Operations such as pattern matching, similarity searching, and forecasting are needed for on-line mining of streaming data.
- *Joins*: Support should be included for multi-stream joins and joins of streams with static meta-data.
- *Windowed queries*: All of the above query types may be constrained to return results inside a window (e.g. the last 24 hours or the last one hundred packets).

3 Data Models and Query Languages for Streams

The above requirements demand specific features to be included in the data models and query languages for data streams. In this section, we survey the proposed models and languages.

3.1 Data Models

A real-time data stream is a sequence of data items that arrive in some order and may be seen only once. Since items may arrive in bursts, a data stream may instead be modeled as a sequence of lists of elements [64]. Individual stream items may take the form of relational tuples or instantiations of objects. In relation-based models (e.g. STREAM [56]), items are transient tuples stored in virtual relations, possibly horizontally partitioned across remote nodes. In object-based models (e.g. COUGAR [10] and Tribeca [61]), sources and item types are modeled as hierarchical data types with associated methods.

In many cases, only an excerpt of a stream is of interest at any given time, giving rise to window models, which may be classified according the the following three criteria [14, 33]:

1. *Direction of movement of the endpoints:* Two fixed endpoints define a *fixed window*, two sliding endpoints (either forward or backward, replacing old items as new items arrive) define a *sliding window*, while one fixed endpoint and one moving endpoint (forward or backward) define a *landmark window*.
2. *Physical vs. logical:* Physical, or *time-based* windows are defined in terms of a time interval, while logical, (also known as *count-based*) windows are defined in terms of the number of tuples.
3. *Update interval:* Eager re-evaluation updates the window upon arrival of each new tuple, while batch processing (lazy re-evaluation) induces a "jumping window". If the update interval is larger than the window size, the result is a series of non-overlapping *tumbling windows* [11].

3.2 Continuous Query Semantics

Assume for simplicity that time is represented as a sequence of integers. Let $A(Q, t)$ be the answer set of a continuous query Q at time t, τ be the current time, and 0 be the starting time. If a continuous query is monotonic, it suffices to re-evaluate the query over newly arrived items and append qualifying tuples to

the result. Thus, the answer set of a monotonic continuous query Q at time τ is [3]:

$$A(Q,\tau) = \bigcup_{t=1}^{\tau} (A(Q,t) - A(Q,t-1)) \cup A(Q,0) \quad (1)$$

In contrast, non-monotonic queries may need to be re-computed from scratch during every re-evaluation, giving rise to the following semantics [3]:

$$A(Q,\tau) = \bigcup_{t=0}^{\tau} A(Q,t) \quad (2)$$

3.3 Stream Query Languages

Three querying paradigms for streaming data have been proposed: relation-based, object-based, and procedural. We briefly describe each group below.

3.3.1 Relation-based Languages

Three proposed relation-based languages are CQL [3, 56], StreaQuel [12, 14], and AQuery [47], each of which has SQL-like syntax and enhanced support for windows and ordering. CQL (Continuous Query Language) is used in the STREAM system, and considers streams and windows to be relations ordered by timestamp. Relation-to-stream operators are provided to convert query results to streams. With these operators, the user may explicitly specify the query semantics, as defined in Equations (1) and (2). Additionally, the sampling rate may be explicitly defined, e.g. ten percent, by following a reference to a stream with the statement `10 % SAMPLE`.

StreaQuel, the query language used in TelegraphCQ, also provides advanced windowing capabilities, and does not require any relation-to-stream operators as it considers all query inputs and outputs to be streams. Each StreaQuel query is followed by a for-loop construct with a variable `t` that iterates over time. The loop contains a `WindowIs` statement that specifies the type and size of the window. Let `S` be a stream and `NOW` be the current time. To specify a sliding window over `S` with size five that should run for fifty time units, the following for-loop may be appended to the query (note that changing the for-loop increment condition to `t=t+5` causes the query to re-execute every five time units):

```
for(t=NOW; t<NOW+50; t++)
   WindowIs(S, t-4, t)
```

AQuery consists of a query algebra and an SQL-based language for ordered data. Table columns are treated as arrays, on which order-dependent operators such as next, previous (abbreviated prev), first, and last may be applied. For example, a continuous query over a stream of stock quotes that reports consecutive price differences of IBM stock may be specified as follows:

```
SELECT price - prev(price)
FROM Trades WHERE company = 'IBM'
```

3.3.2 Object-based Languages

One approach to object-oriented stream modeling is to classify stream elements according to a type hierarchy. This method is used in the Tribeca network monitoring system, which implements Internet protocol layers as hierarchical data types [61]. Another possibility is to model the sources as ADTs, as in the COUGAR sensor database [10]. Each type of sensor is modeled by an ADT, whose interface consists of the sensor's signal processing methods. The proposed query language has SQL-like syntax and also includes a `$every()` clause that indicates the query re-execution frequency; however, few details on the language are available in the published literature (which is why we do not include it in Table 1).

3.3.3 Procedural Languages

An alternative to declarative query languages is to let the user specify the data flow. In the procedural language of the Aurora system [11], users construct query plans via a graphical interface by arranging boxes (corresponding to query operators) and joining them with directed arcs to specify data flow, though the system may later re-arrange, add, or remove operators in the optimization phase. Aurora includes several operators that are not explicitly defined in other languages: *map* applies a function to each item (this operator is also defined in AQuery, where it is called "each"), *resample* interpolates values of missing items within a window, while *drop* randomly drops items if the input rate is too high.

3.3.4 Comments on Query Languages

Table 1 summarizes the proposed streaming query languages. Note that periodic execution refers to allowing the users to specify how often to refresh results. All languages (especially StreaQuel) include extensive support for windowing. In comparison with the list of fundamental query operators in Section 3.3, all required operators except top-k and pattern matching are explicitly defined in all the languages. Nevertheless, user-defined aggregates should make it possible to define pattern-matching functions and extend the language to accommodate future streaming applications. Overall, relation-based

Language/ system	Motivating applications	Allowed inputs	Basic operators	Supported windows			Custom operators?
				type	base	execution	
AQuery	stock quotes, network traffic analysis	sorted relations	relational, "each", order-dependent (first, next, etc.)	fixed, landmark, sliding,	time and count	not discussed in [47]	via "each" operator
Aurora	sensor data	streams only	σ,π,$\cup$,$\bowtie$, group-by, resample, drop, map, window sort	fixed, landmark, sliding	time and count	streaming	via map operator
CQL/ STREAM	all-purpose	streams and relations	relational, relation-to-stream, sample	currently only sliding	time and count	streaming	allowed
StreaQuel/ TelegraphCQ	sensor data	streams and relations	relational	all types	time and count	streaming or periodic	allowed
Tribeca	network traffic analysis	single input stream	σ, π, group-by, union aggregates	fixed, landmark, sliding	time and count	streaming	allows custom aggregates

Table 1: Summary of existing and proposed data stream languages.

languages with additional support for windowing and sequencing appear to be the most popular paradigm at this time.

4 Implementing Streaming Operators

In this section, we discuss issues in implementing streaming operators, including non-blocking behaviour, approximations, and sliding windows. Note that simple operators such as projection and selection (that do not keep state information) may be used in streaming queries without any modifications.

4.1 Non-blocking Operators

Recall that some relational operators are blocking. For instance, prior to returning the next tuple, the Nested Loops Join (NLJ) may potentially scan the entire inner relation and compare each tuple therein with the current outer tuple. Three general techniques exist for unblocking stream operators: windowing, incremental evaluation, and exploiting stream constraints.

Any operator can be unblocked by restricting its range to a finite window, so long as the window fits in memory. To avoid re-scanning the entire window (or stream), streaming operators must be incrementally computable. For example, aggregates such as `AVERAGE` [68] may be incrementally updated by maintaining the cumulative sum and item count. Similarly, a pipelined hash join [66, 69] is a non-blocking join operator, which builds hash tables on-the-fly for each of the participating relations. When a tuple from one of the relations arrives, it is inserted into its table and the other tables are probed for matches. However, an infinite stream may not be buffered in its entirety, so both windowing and incremental evaluation must be applied (see [2] for a discussion of memory requirements of continuous queries).

Another way to unblock query operators is to exploit stream constraints. Schema-level constraints include synchronization among timestamps in multiple streams, clustering (duplicates arrive contiguously), and ordering [9, 37, 56]. Constraints at the data level may take the form of control packets inserted into a stream (referred to as *punctuations* in [64]), which specify any conditions that will hold for all future items, e.g. no other tuples with timestamp smaller than τ will be produced by a given source. There are several open problems concerning punctuations—given an arbitrary query, is there a punctuation that unblocks this query? If so, is there an efficient algorithm for finding this punctuation?

4.2 Approximate Algorithms

If none of the above unblocking conditions are satisfied, compact stream summaries may be stored and approximate queries may be posed over the summaries. This implies a trade-off between accuracy and the amount of memory used to store stream summaries, with an additional restriction that the processing time per item (amortized) should be kept small. We classify approximate algorithms in the infinite stream model according to the method of generating synopses:

- *Counting methods*, used to compute quantiles and frequent item sets, store frequency counts of selected item types (perhaps chosen by sampling) along with error bounds on their true frequencies, e.g. [25, 40, 54].
- *Hashing methods* are generally used with counting or sampling, e.g. for finding frequent items in a stream [27, 30].
- *Sampling methods* compute various aggregates to within a known error bound, but may not be applicable in some cases (e.g. finding a maximum element in a stream). Examples include [25, 27, 34, 54, 55].
- *Sketches* are used in various aggregate queries. Sketching involves taking an inner product of a function of interest (e.g. item frequencies) with a vector of random values chosen from some distribution with a known expectation. Some examples include [1, 15, 20, 26, 29, 33, 37].
- *Wavelet transforms* that reduce the underlying signal to a small set of coefficients have been proposed to approximate aggregates over infinite streams, e.g. [32, 37, 41].

4.3 Data Stream Mining

On-line stream mining operators must be incrementally updatable without making multiple passes over the data. Recent results in (approximate) algorithms for on-line stream mining include computing stream signatures and representative trends [21], decision trees [44], forecasting [71], k-medians clustering [16, 42], nearest neighbour queries [46], and regression analysis [18]. A comprehensive discussion of similarity detection, pattern matching, and forecasting in sensor data mining may be found in [28].

4.4 Sliding Window Algorithms

Many infinite stream algorithms do not have obvious counterparts in the sliding window model. For instance, while computing the maximum value in an infinite stream is trivial, doing so in a sliding window of size N requires $\Omega(N)$ space—consider a sequence of non-increasing values, in which the maximum item is always expired when the window moves forward. Thus, the fundamental problem is that as new items arrive, old items must be simultaneously evicted.

In addition to windowed sampling [7], a possible solution to computing sliding window queries in sublinear space is to divide the window into small portions (called *basic windows* in [72]) and only store a synopsis and a timestamp for each portion. When the timestamp of the oldest basic window expires, its synopsis is removed, a fresh window is added to the front, and the aggregate is incrementally recomputed. This method may be used to compute correlations between streams [72], find frequently appearing items [24], and compute various aggregates [8, 23, 35]. However, some window statistics may not be incrementally computable from a set of synopses.

The symmetric hash join [69] and an analogous symmetric NLJ may be extended to operate over two [45] or more [38] sliding windows by periodically scanning the hash tables (or whole windows) and removing stale items. Interesting trade-offs appear in that large hash tables are expensive to maintain if tuple expiration is performed too frequently [38].

5 Continuous Query Processing and Optimization

We now discuss problems related to processing and optimizing continuous queries. In what follows, we outline emerging research in cost metrics, query plans, quality-of-service guarantees, and distributed optimization of streaming queries.

5.1 Cost Metrics and Statistics

Traditional cost metrics do not apply to (possibly approximate) continuous queries over infinite streams, where processing cost per-unit-time is more appropriate [45]. Below, we list possible cost metrics for streaming queries along with necessary statistics that must be maintained.

- *Accuracy and reporting delay vs. memory usage*: Sampling and load shedding [62] may be used to decrease memory usage by increasing the error. It is necessary to know the accuracy of each operator as a function of the available memory, and how to combine such functions to obtain the overall accuracy of a plan. Furthermore, batch processing [6] may be done instead of re-evaluating a query whenever a new item arrives.
- *Output rate*: If the stream arrival rates and output rates of query operators are known, it is possible to optimize for the highest output rate or to find a plan that takes the least time to output a given number of tuples [67].
- *Power usage*: In a wireless network of battery-operated sensors, energy consumption may be minimized if each sensor's power consumption

characteristics (when transmitting and receiving) are known [51, 70].

5.2 Continuous Query Plans

In relational DBMSs, all operators are pull-based: an operator requests data from one of its children in the plan tree only when needed. In contrast, stream operators consume data pushed to the system by the sources. One approach to reconcile these differences, as considered in Fjords [49] and STREAM [6], is to connect operators with queues, allowing sources to push data into a queue and operators to retrieve data as needed. Since queues may overflow, operators should be scheduled so as to minimize queue sizes and queuing delays [5]. Another challenge in continuous query plans deals with supporting historical queries. Designing disk-based data structures and indices to exploit access patterns of stream archives is an open problem [12].

5.3 Processing Multiple Queries

Two approaches have been proposed to execute similar continuous queries together: sharing query plans [17] and indexing query predicates [13, 52]. In the former, queries belonging to the same group share a plan, which produces the union of the results needed by each query in the group. A final selection is then applied to the shared result set. Problems include dynamic re-grouping as new queries are added to the system, and shared evaluation of windowed joins with various window sizes [43].

In the indexing approach, query predicates are stored in a table. When a new tuple arrives for processing, its attribute values are extracted and looked up in the query table to see which queries are satisfied by this tuple. Data and queries are treated as duals, reducing query processing to a multi-way join of the predicate table with the data tables. The indexing approach works well for queries with simple boolean predicates, but is currently not applicable to, e.g. windowed aggregates [13].

5.4 Query Optimization

5.4.1 Query Rewriting

A useful rewriting technique in relational databases deals with re-ordering a sequence of binary joins in order to minimize a particular cost metric. There has been some preliminary work in join ordering for data streams in the context of the rate-based model [67] and in main-memory window joins [38]. In general, each of the query languages outlined in Section 3 introduces some new rewritings, e.g. commutativity of selections and projections over sliding windows [3, 14].

5.4.2 Adaptivity

The cost of a query plan may change for three reasons: change in processing time of an operator, change in selectivity of a predicate, and change in the arrival rate of a stream [4]. Consequently, instead of maintaining a rigid tree-structured query plan, the Eddies approach (introduced in [4], extended to multi-way joins in [58], and applied to continuous queries in [13, 52]) performs scheduling of each tuple separately by routing it through the operators that make up the query plan. In effect, the query plan is dynamically re-ordered to match current system conditions. This is accomplished by tuple routing policies that attempt to discover which operators are fast and selective, and those operators are scheduled first. There is, however, an important trade-off between the resulting adaptivity and the overhead required to route each tuple separately.

5.5 Distributed Query Processing

In sensor networks, Internet traffic analysis, and Web usage logs, multiple data streams arrive from remote sources. Distributed optimization strategies aim at decreasing communication costs by re-ordering query operators across sites [19, 59] and performing simple query functions (e.g. filtering or aggregation) locally at a sensor or a network router [22, 50, 53, 70]. For example, if each node pre-aggregates its results by sending to the central node the sum and count of its values, the co-ordinator may then take the cumulative sum and cumulative count, and compute the overall average. A similar technique involves sending updates to the central node only if new data values differ significantly from previously reported values [57].

Some optimization techniques have been designed specifically for ad-hoc wireless sensor networks in order to decrease communication costs, extend battery life, and deal with poor wireless connectivity [50, 53]. These techniques make use of the fact that query dissemination and result collection in a wireless sensor network proceed along a routing tree (or a DAG) via a shared wireless channel. For example, if a sensor reports its maximum local value x in response to a `MAX` query, a neighbouring sensor that overhears this transmission need not respond if its local maximum is smaller than x. Moreover, a sensor could broadcast redundant copies of its maximum value to all of its neighbours in order to minimize the chances of

packet loss (at a cost of increased bandwidth and battery usage). However, this does not work for other aggregates such as `SUM` and `COUNT`, as duplicate values would contaminate the result. In these cases, a sensor may "split" its local sum and send partial sums to each of its neighbours. Even if one packet is lost, the remainder of the sum should still reach the root.

6 Conclusions

Designing an effective data stream management system requires extensive modifications of nearly every part of a traditional database, creating many interesting database problems such as adding time, order, and windowing to data models and query languages, implementing approximate operators, combining push-based and pull-based operators in query plans, adaptive query re-optimization, and distributed query processing. Recent interest in these problems has generated a number of academic projects. There exist at least the following systems:

- Aurora [11, 19] is a workflow-oriented system that allows users to build query plans by arranging boxes (operators) and arrows (data flow among operators). `http://www.cs.brown.edu/research/aurora`.
- COUGAR [10, 70] is a sensor database that models sensors as ADTs and their output as time series. `http://www.cs.cornell.edu/database/cougar`.
- Gigascope [22] is a distributed network monitoring architecture that proposes pushing some query operators to the sources (e.g. routers).
- NiagaraCQ [17] is a continuous query system that allows continuous XML-QL queries to be posed over dynamic Web content. `http://www.cs.wisc.edu/niagara`.
- OpenCQ [48] is another continuous query system for monitoring streaming Web content. Its focus is on scalable event-driven query processing. `http://disl.cc.gatech.edu/CQ`
- StatStream [72] is a stream monitoring system designed to compute online statistics across many streams. `http://cs.nyu.edu/cs/faculty/shasha/papers/statstream.html`.
- STREAM [56] is an all-purpose relation-based system with an emphasis on memory management and approximate query answering. `http://www-db.stanford.edu/stream`.
- TelegraphCQ [12] is a continuous query processing system that focuses on shared query evaluation and adaptive query processing `http://telegraph.cs.berkeley.edu`.
- Tribeca [61] is an early on-line Internet traffic monitoring tool.

References

[1] N. Alon, Y. Matias, M. Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proc. ACM Symp. on Theory of Computing*, 1996, pp. 20–29.

[2] A. Arasu, B. Babcock, S. Babu, J. McAlister, J. Widom. Characterizing Memory Requirements for Queries over Continuous Data Streams. In *Proc. ACM Symp. on Principles of Database Systems*, 2002, pp. 221–232.

[3] A. Arasu, S. Babu, J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. Technical Report, Nov. 2002. `dbpubs.stanford.edu:8090/pub/2002-57`.

[4] R. Avnur, J. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proc. ACM Int. Conf. on Management of Data*, 2000, pp. 261–272.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. To appear in *Proc. ACM Int. Conf. on Management of Data*, June 2003.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Streams. In *Proc. ACM Symp. on Principles of Database Systems*, 2002, pp. 1–16.

[7] B. Babcock, M. Datar, R. Motwani. Sampling from a Moving Window over Streaming Data. In *Proc. SIAM-ACM Symp. on Discrete Algorithms*, 2002, pp. 633–634.

[8] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan. Maintaining Variance and k-Medians over Data Stream Windows. To appear in *Proc. ACM Symp. on Principles of Database Systems*, June 2003.

[9] S. Babu, J. Widom. Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams. Technical Report, Nov. 2002. `dbpubs.stanford.edu:8090/pub/2002-52`.

[10] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. In *Proc. Int. Conf. on Mobile Data Management*, 2001, pages 3–14.

[11] D. Carney, U. Cetinternel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. Monitoring streams—A New Class of Data Management Applications. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 215–226.

[12] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M. Shah. TelegraphCQ: Continuous Dataflow Processing for

an Uncertain World. In *Proc. Conf. on Innovative Data Syst. Res*, 2003, pp. 269–280.

[13] S. Chandrasekaran, M. J. Franklin. Streaming Queries over Streaming Data. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 203–214.

[14] S. Chandrasekaran, S. Krishnamurthy, S. Madden, A. Deshpande, M. J. Franklin, J. M. Hellerstein, M. Shah. Windows Explained, Windows Expressed. 2003. `www.cs.berkeley.edu/~sirish/research/streaquel.pdf`.

[15] M. Charikar, K. Chen, M. Farach-Colton. Finding frequent items in data streams. In *Proc. Int. Colloquium on Automata, Languages and Programming*, 2002, pp. 693–703.

[16] M. Charikar, L. O'Callaghan, R. Panigrahy. Better Streaming Algorithms for Clustering Problems. To appear in *Proc. ACM Symp. on Theory of Computing*, June 2003.

[17] J. Chen, D. DeWitt, F. Tian, Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. ACM Int. Conf. on Management of Data*, 2000, pp. 379–390.

[18] Y. Chen, G. Dong, J. Han, B. W. Wah, J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 323–334.

[19] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. Zdonik. Scalable Distributed Stream Processing. In *Proc. Conf. on Innovative Data Syst. Res*, 2003.

[20] G. Cormode, M. Datar, P. Indyk, S. Muthukrishnan. Comparing Data Streams Using Hamming Norms (How to Zero In). In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 335–345.

[21] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, F. Smith. Hancock: A Language for Extracting Signatures from Data Streams. In *Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining*, 2000, pp. 9–17.

[22] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, O. Spatscheck. GigaScope: High Performance Network Monitoring with an SQL Interface. In *Proc. ACM Int. Conf. on Management of Data*, 2002, p. 623.

[23] M. Datar, A. Gionis, P. Indyk, R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *Proc. SIAM-ACM Symp. on Discrete Algorithms*, 2002, pp. 635–644

[24] D. DeHaan, E. D. Demaine, L. Golab, A. Lopez-Ortiz, J. I. Munro. Towards Identifying Frequent Items in Sliding Windows. Technical Report, March 2003. `db.uwaterloo.ca/~lgolab/frequent.pdf`.

[25] E. Demaine, A. Lopez-Ortiz, J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proc. European Symp. on Algorithms*, 2002, pp. 348–360.

[26] A. Dobra, M. Garofalakis, J. Gehrke, R. Rastogi. Processing Complex Aggregate Queries over Data Streams. In *Proc. ACM Int. Conf. on Management of Data*, 2002, pp. 61–72.

[27] C. Estan, G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, 2001, pp. 75–80.

[28] C. Faloutsos. Sensor Data Mining: Similarity Search and Pattern Analysis. Tutorial in *Proc. Int. Conf. on Very Large Data Bases*, 2002.

[29] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan. An Approximate L1-Difference Algorithm for Massive Data Streams. In *Proc. Symp. on Foundations of Computer Science*, 1999. pp. 501–511.

[30] P. Flajolet, G. N. Martin. Probabilistic Counting. In *Proc. Symp. on Foundations of Computer Science*, 1983, pp. 76–82, 1983.

[31] M. Garofalakis, J. Gehrke, R. Rastogi. Querying and Mining Data Streams: You Only Get One Look. Tutorial in *ACM Int. Conf. on Management of Data*, 2002.

[32] M. Garofalakis, P. Gibbons. Wavelet Synopses with Error Guarantees. In *Proc. ACM Int. Conf. on Management of Data*, 2002, pp. 476–487.

[33] J. Gehrke, F. Korn, D. Srivastava. On Computing Correlated Aggregates Over Continual Data Streams. In *Proc. ACM Int. Conf. on Management of Data*, 2001, pp. 13–24.

[34] P. Gibbons, S. Tirthapura. Estimating Simple Functions on the Union of Data Streams. In *Proc. ACM Symp. on Parallel Algorithms an Architectures*, 2001, pp. 281–291.

[35] P. Gibbons, S. Tirthapura. Distributed Streams Algorithms for Sliding Windows. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 2002, pp. 63–72.

[36] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. J. Strauss. QuickSAND: Quick Summary and Analysis of Network Data. Technical Report, Dec. 2001. `citeseer.nj.nec.com/gilbert01quicksand.html`

[37] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. J. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proc. Int. Conf. on Very Large Data Bases*, 2001, pp. 79–88.

[38] L. Golab, M. T. Özsu. Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. Technical Report, Feb. 2003. `db.uwaterloo.ca/~ddbms/publications/stream/multijoins.pdf`.

[39] L. Golab, M. T. Özsu. Data Stream Management Issues – A Survey. Technical Report, Apr. 2003. `db.uwaterloo.ca/~ddbms/publications/stream/streamsurvey.pdf`.

[40] J. Greenwald, F. Khanna. Space Efficient On-Line Computation of Quantile Summaries. In *Proc. ACM Int. Conf. on Management of Data*, 2001, pp. 58–66.

[41] S. Guha, P. Indyk, S. Muthukrishnan, M. Strauss. Histogramming Data Streams with Fast Per-Item Processing. In *Proc. Int. Colloquium on Automata, Languages and Programming*, 2002, pp. 681–692.

[42] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan. Clustering Data Streams. In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 359–366.

[43] M. A. Hammad, M. J. Franklin, W. G. Aref, A. K. Elmagarmid. Scheduling for shared window joins over data streams. Submitted for publication, Feb. 2003.

[44] G. Hulten, L. Spencer, P. Domingos. Mining Time-Changing Data Streams. In *Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining*, 2001, pp. 97–106.

[45] J. Kang, J. Naughton, S. Viglas. Evaluating Window Joins over Unbounded Streams. To appear in *Proc. Int. Conf. on Data Engineering*, 2003.

[46] F. Korn, S. Muthukrishnan, D. Srivastava. Reverse Nearest Neighbor Aggregates over Data Streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 814–825.

[47] A. Lerner, D. Shasha. AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments. Technical Report, March 2003. `csdocs.cs.nyu.edu/Dienst/Repository/2.0/Body/ncstrl.nyu_cs%2fTR2003-836/pdf`.

[48] L. Liu, C. Pu, W. Tang. Continual Queries for Internet-Scale Event-Driven Information Delivery. In *IEEE Trans. Knowledge and Data Eng.*, 11(4): 610–628, 1999.

[49] S. Madden, M. J. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. In *Proc. Int. Conf. on Data Engineering*, 2002, pp. 555–566.

[50] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. Symp. on Operating Systems Design and Implementation*, 2002.

[51] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong. The Design of an Acquisitional Query Processor For Sensor Networks. To appear in *Proc. ACM Int. Conf. on Management of Data*, June 2003.

[52] S. Madden, M. Shah, J. Hellerstein, V. Raman. Continuously Adaptive Continuous Queries Over Streams. In *Proc. ACM Int. Conf. on Management of Data*, 2002, pp. 49–60.

[53] S. Madden, R. Szewczyk, M. J. Franklin, D. Culler. Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 49–58.

[54] G. S. Manku, R. Motwani. Approximate Frequency Counts over Data Streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 346–357.

[55] G.S. Manku, S. Rajagopalan, B.G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *Proc. ACM Int. Conf. on Management of Data*, 1999, pp. 251–262.

[56] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, R. Varma. Query Processing, Approximation, and Resource Management in a Data Stream Management System. In *Proc. Conf. on Innovative Data Syst. Res*, 2003, pp. 245–256.

[57] C. Olston, J. Jiang, J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. To appear in *Proc. ACM Int. Conf. on Management of Data*, June 2003.

[58] V. Raman, A. Deshpande, J. Hellerstein. Using State Modules for Adaptive Query Processing. To appear in *Proc. Int. Conf. on Data Engineering*, 2003.

[59] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, M. J. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. To appear in *Proc. Int. Conf. on Data Engineering*, 2003.

[60] Stream Query Repository, `www-db.stanford.edu/stream/sqr`.

[61] M. Sullivan, A. Heybey. Tribeca: A System for Managing Large Databases of Network Traffic. In *Proc. USENIX Annual Technical Conf.* , 1998.

[62] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, M. Stonebraker. Load Shedding in a Data Stream Manager. Technical Report, Feb. 2003. `www.cs.brown.edu/~tatbul/papers tatbul_tr.pdf`.

[63] Traderbot, `www.traderbot.com`.

[64] P. Tucker, D. Maier, T. Sheard, L. Fegaras. Enhancing relational operators for querying over punctuated data streams. 2002. `www.cse.ogi.edu/dot/niagara/pstream/punctuating.pdf`.

[65] P. Tucker, T. Tufte, V. Papadimos, D. Maier. NEXMark—a Benchmark for Querying Data Streams. 2002. `www.cse.ogi.edu/dot/niagara/pstream/nexmark.pdf`.

[66] T. Urhan, M. J. Franklin. XJoin: A Reactively-Scheduled Pipelined Join Operator. In *IEEE Data Engineering Bulletin*, 23(2):27–33, June 2000.

[67] S. Viglas and J. Naughton. Rate-Based Query Optimization for Streaming Information Sources. In *Proc. ACM Int. Conf. on Management of Data*, 2002, pp. 37–48.

[68] H. Wang, C. Zaniolo. ATLaS: A Native Extension of SQL for Data Mining and Stream Computations. `citeseer.nj.nec.com/551711.html`.

[69] A. Wilschut, P. Apers. Dataflow query execution in a parallel main-memory environment. In *Proc. Int. Conf. Parallel and Distributed Information Systems*, 1991, pp. 68–77.

[70] Y. Yao and J. Gehrke. Query Processing for Sensor Networks. In *Proc. Conf. on Innovative Data Syst. Res*, 2003, pp. 233–244.

[71] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, A. Biliris. On-Line Data Mining for Co-Evolving Time Sequences. In *Proc. Int. Conf. on Data Engineering*, 2000, pp. 13–22.

[72] Y. Zhu, D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proc. Int. Conf. on Very Large Data Bases*, 2002, pp. 358–369.