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.

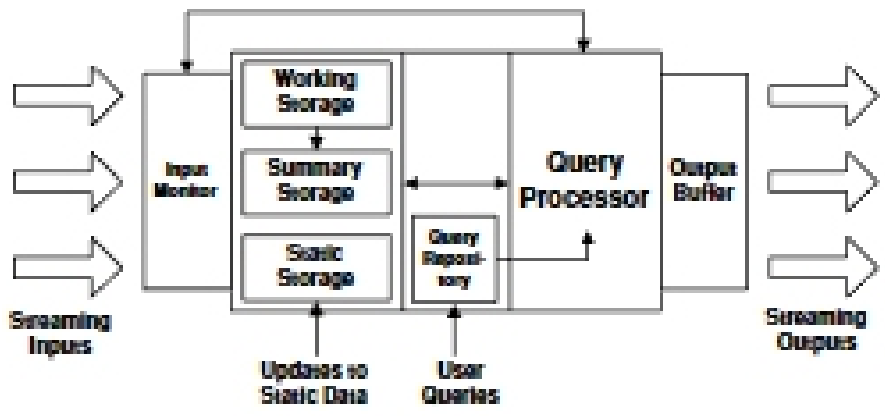

Figure 1: Abstract reference architecture for a data stream management system.

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