StreaMon: An Adaptive Engine for Stream Query Processing[*]

Shivnath Babu
Stanford University
shivnath@cs.stanford.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

ABSTRACT

StreaMon is the adaptive query processing engine of the *STREAM* prototype *Data Stream Management System* (*DSMS*) [4]. A fundamental challenge in many DSMS applications (e.g., network monitoring, financial monitoring over stock tickers, sensor processing) is that conditions may vary significantly over time. Since queries in these systems are usually long-running, or *continuous* [4], it is important to consider *adaptive* approaches to query processing. Without adaptivity, performance may drop drastically as stream data and arrival characteristics, query loads, and system conditions change over time.

StreaMon uses several techniques to support adaptive query processing [1, 2, 3]; we demonstrate three of them:

- Reducing run-time memory requirements for continuous queries by exploiting stream data and arrival patterns.
- Adaptive join ordering for pipelined multiway stream joins, with strong quality guarantees.
- Placing subresult caches adaptively in pipelined multiway stream joins to avoid recomputation of intermediate results.

1. INTRODUCTION

STREAM is a relational DSMS that supports continuous queries specified in a rich declarative language called *CQL* [4]. Each CQL query is compiled into a query plan that runs continuously. As the plan executes, it is optimized adaptively by the *StreaMon* engine. Our approach is to generate a straightforward initial query plan, which then adapts automatically to a better initial plan, and continues to adapt as conditions change. STREAM supports an interactive graphical interface for visualizing run-time plan and system behavior. Using this interface, users can visualize and control plan adaptivity.

Consider a sample continuous query from a network monitoring application for an Internet Service Provider [3]:

> *Monitor the total traffic from an incoming link L_1 that went through an internal link L_2 on to an outgoing link L_3 over the last 10 minutes.*

Data collection devices on the links feed three streams, which for convenience we also denote L_1, L_2, and L_3. Each stream tuple contains a packet identifier *pid* and the packet's *size*. Figure 1 shows a multiway join for executing this query. Each stream feeds a 10-minute sliding window, W_i for L_i, with a hash index on *pid*. When a tuple t is inserted into W_1, the other two windows are probed with t.*pid* in some order, e.g., the order W_3, W_2 is used in Figure 1. If both windows contain a tuple matching t.*pid* (*pid* is unique), then the joined tuple is sent as an insertion to the aggregation operator *sum(size)*. Otherwise, processing on t stops at the first window that does not contain a matching tuple. Similar processing occurs for deletions from W_1, and for insertions and deletions to the other two windows (not shown in the figure).

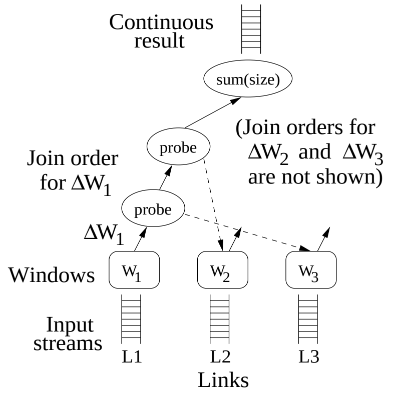

Figure 1: Plan for example query

Streams L_1–L_3 exhibit some interesting properties. First, the monitored packets flow through link L_1 to L_2 to L_3. Thus, a tuple corresponding to a specific *pid* appears in stream L_1 first, then a joining tuple may appear in stream L_2, and lastly in stream L_3. Second, if the current latency of the network between links L_1 and L_2 and between links L_2 and L_3 is bounded by d_{12} and d_{23} respectively, then a packet that flows through links L_1, L_2, and L_3 will appear in stream L_2 no later than d_{12} time units after it appears in stream L_1, and in L_3 no later than d_{23} time units after it appears in L_2. StreaMon can detect and exploit such properties to reduce the overall memory requirement for windows W_1–W_3 significantly.

In addition to memory requirements, stream characteristics affect the cost of the join plan used for each stream. If almost none of the packets on L_1 get to L_3, but a large fraction of the packets on L_2 pass through L_3, then the join order W_1, W_2 for tuples in W_3 can be up to twice as efficient as the order W_2, W_1. If the arrival rate of stream L_1 is much higher than that of streams L_2 and L_3, then it may be best to keep $W_2 \bowtie W_3$ materialized, to avoid excessive recomputation.

Finally, routing paths in the network may change, e.g., in response to congestion and failures, so the characteristics of these streams, and therefore the efficiency of the join plans, are expected to vary over time.

[*]This work was supported by the National Science Foundation under grants IIS-0118173 and IIS-9817799.

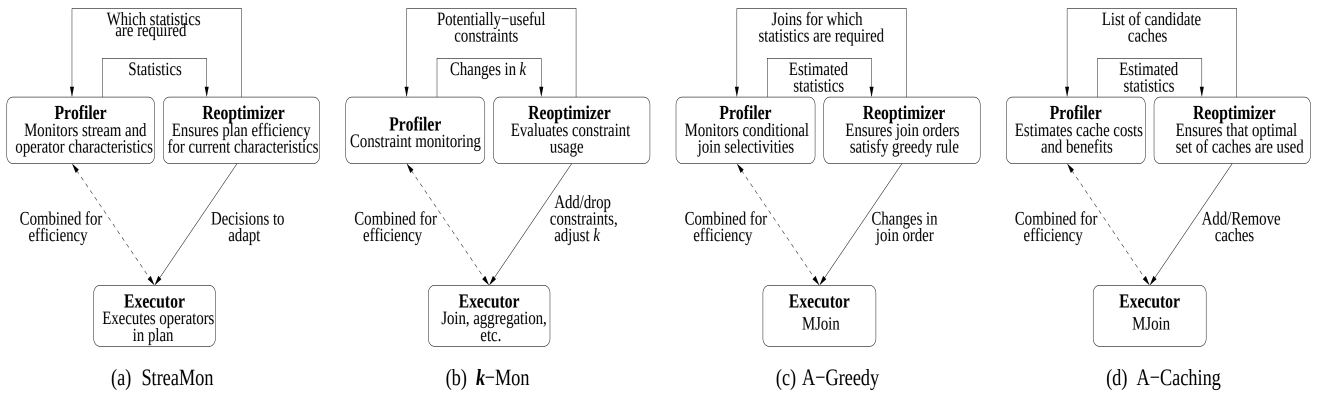

Figure 2: Basic structure of StreaMon, k-Mon, A-Greedy, and A-Caching

2. STREAMON

The simple network monitoring example isolates some of the important challenges in stream processing: Stream and query plan characteristics can change over the lifetime of a continuous query, and stream characteristics can be correlated. Exploiting current stream characteristics during query processing can reduce memory requirements and improve throughput and response times significantly. StreaMon addresses these challenges.

StreaMon has three generic components as shown in Figure 2(a): an *Executor*, which runs query plans to produce results, a *Profiler*, which collects and maintains statistics about stream and plan characteristics, and a *Reoptimizer*, which ensures that the plans and memory usage are the most efficient for current input characteristics. We demonstrate three features of StreaMon:

Adaptive Memory Minimization: Many stream properties useful for memory reduction in continuous queries can be captured, either individually or in combination, by a set of basic *constraints*: *many-one joins*, *stream-based referential integrity*, *ordering*, and *clustering* [3]. It is unreasonable to expect streams to satisfy stringent constraints at all times, due to variability in data generation, network load, scheduling, and other factors, so we developed the notion of k-*constraints*: $k \geq 0$ is an *adherence parameter* capturing the degree to which a stream or joining pair of streams adheres to the strict interpretation of the constraint. (The constraint holds with its strict interpretation when $k = 0$.) For example, k-*ordering* specifies that out-of-order stream elements are no more that k elements apart. StreaMon can detect useful k-constraints in streams and exploit them to reduce memory requirements for stateful operators like aggregation and join. Figure 2(b) shows k-*Mon*, the part of StreaMon that handles k-constraints. k-Mon's Profiler monitors input streams and informs the Reoptimizer of potentially-useful constraints, and whenever relevant k values change. The Reoptimizer tells stateful operators to start or stop using a constraint, or to adjust the value of k used. k-Mon is described in [3].

Adaptive Join Ordering: The pipelined multiway join algorithm in Figure 1 is called *MJoin* [5]. MJoin requires a join order for each input stream, used to join new tuples in that stream with the windows of the other streams. StreaMon supports an algorithm called *A-Greedy* (for *Adaptive-Greedy*) that maintains join orders adaptively for MJoins; see Figure 2(c). A-Greedy's Profiler monitors conditional join selectivities and its Reoptimizer (re)orders joins to minimize overall work in current conditions. In stable conditions, the orderings converged on by A-Greedy are equivalent to those selected by a static Greedy algorithm that is provably within a cost factor < 4 of optimal. In practice, the Greedy algorithm, and therefore A-Greedy, nearly always finds the optimal orderings. StreaMon also supports three variants of A-Greedy that make different tradeoffs among provable convergence to good join orders, run-time overhead, and speed of adaptivity. A-Greedy and its variants are described in [1].

Adaptive Caching for Joins: In conjunction with adaptive join ordering, StreaMon supports an algorithm called *A-Caching* (for *Adaptive-Caching*) that adds and removes *subresult caches* adaptively in MJoins to avoid recomputing intermediate results, when it is beneficial to do so [2]. With A-Caching, our pipelined multiway joins can adapt over the entire spectrum between stateless MJoins and cache-rich join trees, as stream and system conditions change. As shown in Figure 2(d), A-Caching's Profiler monitors cache cost and benefits, and its Reoptimizer selects caches to use, allocating memory to caches dynamically. A-Caching is described in [2].

We are now applying the StreaMon approach to make more aspects of the DSMS adaptive, e.g., plan sharing and operator scheduling. We are also addressing the problem of balancing query execution against profiling and reoptimization overhead for optimal performance in an adaptive setting.

3. ACKNOWLEDGMENTS

We are grateful to our collaborators in StreaMon: Rajeev Motwani, Kamesh Munagala, and Utkarsh Srivastava.

4. REFERENCES

[1] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, June 2004.

[2] S. Babu, K. Munagala, J. Widom, and R. Motwani. Adaptive caching for continuous queries. Technical report, Stanford University Database Group, Mar. 2004. Available at http://dbpubs.stanford.edu/pub/2004-14.

[3] S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. Technical report, Stanford University Database Group, Nov. 2002. Available at http://dbpubs.stanford.edu/pub/2002-52.

[4] R. Motwani, J. Widom, et al. Query processing, approximation, and resource management in a data stream management system. In *Proc. First Biennial Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[5] S. Viglas, J. Naughton, and J. Burger. Maximizing the output rate of multi-join queries over streaming information sources. In *Proc. of the 2003 Intl. Conf. on Very Large Data Bases*, Sept. 2003.