

CS848 Presentation Report

STREAM System

Jiang Wu
School of Computer Science
University of Waterloo

February 8, 2006

1 Introduction

STREAM is the Data Stream Management System (DSMS) developed at Stanford University. It supports a large class of declarative continuous queries over continuous data streams as well as relational databases.

This report is a introduction to STREAM. It is based on the publications on STREAM [1, 2, 3, 4, 5], and focuses on semantics and implementation of continuous query, and techniques used in STREAM to reduce CPU and memory usage. The outline of the report is as follows:

section 2: Continuous Query Language (CQL)

- semantics
- operators

section 3: Implementation of CQL

- internal data representation
- query plan, query execution

section 4: Reduce CPU usage by load-shedding

section 5: Reduce the memory overhead

- Synopsis sharing

- Operator scheduling
- Exploiting constraints

section 6: Conclusion

2 CQL

2.1 Semantics

STREAM defined a formal abstract semantics for continuous queries. The abstract semantics is based on two data types, streams and relations, which are defined using a discrete, ordered time domain $T = \{0, 1, \dots\}$.

A stream S is an unbounded bag (multiset) of pairs $\langle s, t \rangle$, where s is a tuple and $t \in T$ is the timestamp that denotes the logical arrival time of tuple s on stream S .

A relation R is a time-varying bag of tuples. The bag of tuples at time $t \in T$ is denoted $R(t)$, and $R(t)$ is called an instantaneous relation.

S up to t denotes the bag of elements in stream S with timestamps $\leq t$:

$$\{\langle s, t' \rangle \in S : t' \leq t\}.$$

S at t denotes the bag of elements of S with timestamp t :

$$\{\langle s, t' \rangle \in S : t' = t\}.$$

R at t denotes the instantaneous relation $R(t)$.

R up to t denotes the collection of instantaneous relations $R(0), \dots, R(t)$.

Semantics for continuous queries is based on three classes of operators over streams and relations:

1. stream-to-relation operators that produce a relation from a stream

A stream-to-relation operator takes a stream S as input and produces a relation R as output with the same schema as S . At any instant t , $R(t)$ should be computable from S up to t .

2. relation-to-relation operators that produce a relation from one or more other relations

A relation-to-relation operator takes one or more relations R_1, \dots, R_n as input and produces a relation R as output. At any instant t , $R(t)$ should be computable from $R_1(t), \dots, R_n(t)$.

3. relation-to-stream operators that produce a stream from a relation

A relation-to-stream operator takes a relation R as input and produces a stream S as output with the same schema as R . At any instant t , S at t should be computable from R up to t .

Continuous Semantics is defined as follows: consider a query Q that is any type-consistent composition of operators from the above three classes. Suppose the set of all inputs to the innermost (leaf) operators of Q are streams S_1, \dots, S_n and relations R_1, \dots, R_m , for the result of continuous query Q at a time t , which denotes the result of Q once all inputs up to t are available. There are two cases:

1. The outermost (topmost) operator in Q is relation-to-stream, producing a stream S . The result of Q at time t is S up to t , produced by recursively applying the operators comprising Q to streams $S_1 \dots, S_n$ up to t and relations R_1, \dots, R_m up to t .
2. The outermost (topmost) operator in Q is stream-to-relation or relation-to-relation, producing a relation R . The result of Q at time t is $R(t)$, produced by recursively applying the operators comprising Q to streams $S_1 \dots, S_n$ up to t and relations R_1, \dots, R_m up to t .

2.2 CQL Operators

2.2.1 Stream-to-Relation Operators

In STREAM, all stream-to-relation operators in CQL are based on the concept of a sliding window over a stream: a window that at any point of time contains a historical snapshot of a finite portion of the stream. There are three classes of sliding window operators in CQL: time-based, tuple-based, and partitioned, defined as follows.

- Time-based sliding windows

A time-based sliding window on a stream S takes a time-interval T as a parameter and is specified by following the reference to S with [Range T]. The output relation R of S [Range T] is defined as:

$$R(t) = \{s | \langle s, t' \rangle \in S \wedge (t' \leq t) \wedge (t' \geq \max\{t - T, 0\})\}.$$

- Tuple-based sliding windows

A tuple-based sliding window on a stream S takes a positive integer N as a parameter and is specified by following the reference to S in the query with [Rows N]. For the output relation R of S [Rows N], $R(t)$ consists of the N tuples of S with the largest timestamps $\leq t$.

- Partitioned windows

A partitioned sliding window on a stream S takes a positive integer N and a subset $\{A_1, \dots, A_k\}$ of S 's attributes as parameters. It is specified by following the reference to S in the query with [Partition By A_1, \dots, A_k Rows N]. This window logically partitions S into different substreams based on equality of attributes $\{A_1, \dots, A_k\}$ (similar to SQL Group By), computes a tuple-based sliding window of size N independently on each substream, then takes the union of these windows to produce the output relation.

2.3 Relation-to-Stream Operators

CQL has three relation-to-stream operators: Istream, Dstream, and Rstream. In the following definitions, operators $\cup, \times, -$ are assumed to be the bag versions.

- Istream (for insert stream) applied to relation R contains a stream element $\langle s, t \rangle$ whenever tuple s is in $R(t) - R(t - 1)$.

$$Istream(R) = \bigcup_{r \geq 0} ((R(t) - R(t - 1)) \times t)$$

- Dstream (for delete stream) applied to relation R contains a stream element $\langle s, t \rangle$ whenever tuple s is in $R(t - 1) - R(t)$.

$$Dstream(R) = \bigcup_{r > 0} ((R(t - 1) - R(t)) \times t)$$

- Rstream (for relation stream) applied to relation R contains a stream element $\langle s, t \rangle$ whenever tuple s is in R at time t . Formally:

$$Rstream(R) = \bigcup_{r \geq 0} (R(t) \times t)$$

2.4 Relation-to-Relation Operators

The relation-to-relation operators in CQL are derived from traditional relational queries expressed in SQL. Anywhere a traditional relation is referenced in a SQL query, a relation can be referenced in CQL.

3 CQL Implementation

3.1 Internal Representation of Streams and Relations

Within STREAM, the two fundamental data types in CQL, streams and relations, are unified as sequences of tagged and timestamped tuples. The tag is insertion (+) or deletion (-). The tag-timestamp-tuple triples are referred to as elements. Streams only include + elements, while relations may include both + and - elements to capture the changing relation state over time.

3.2 Query Plans

When a continuous query specified in CQL is registered with the STREAM system, a query plan is compiled from it. Query plans are composed of 1.) operators, which perform the actual processing, 2.) queues, which buffer elements as they move between operators, and 3.) synopses, which store operator state.

3.2.1 Operators

Each query plan operator reads from one or more input queues, processes the input based on its semantics, and writes any output to an output queue. Individual operators may materialize their relational inputs in synopses if such state is useful.

3.2.2 Queues

A queue in a query plan connects its “producing” plan operator O_P to its “consuming” operator O_C . At any time a queue contains a collection of elements representing a portion of a stream or relation. The elements that O_P produces are inserted into the queue and buffered there until they are processed by O_C .

3.2.3 Synopsis

Logically, a synopsis belongs to a specific plan operator, storing state that may be required for future evaluation of that operator. The most common use of a synopsis in our system is to materialize the current state of a (derived) relation, such as the contents of a sliding window or the relation produced by a subquery. Synopses also may be used to store a summary of the tuples in a stream or relation for approximate query answering.

3.3 Query Plan Execution

When a query plan is executed, a scheduler selects operators in the plan to execute in turn. The semantics of each operator depends only on the timestamps of the elements it processes, not on system or wall-clock time. Thus, the order of execution has no effect on the data in the query result, although it can affect other properties such as latency and resource utilization.

4 Reduce CPU Usage

STREAM processes continuous queries over data streams that are often bursty, with unpredictable peaks during which the load may exceed available system resources. The data arrival rate may be so high that there is insufficient CPU time to process each stream element. STREAM solves this problem by load shedding - dropping elements from the query plan before they are processed, therefore to save the CPU time.

The class of continuous monitoring queries that are considered in STREAM are sliding window aggregate queries over continuous data streams. STREAM introduces shedding operators, or load shedders, at various points in the query plan. Each load shedder is parameterized by a sampling rate p . The load shedder flips a coin for each tuple that passes through it. With probability p , the tuple is passed on to the next operator, and with probability $1 - p$, the tuple is discarded.

Load shedding incurs inaccuracy in the query result. In STREAM load shedding is formalized as an optimization problem with the objective of minimizing query inaccuracy within the limits imposed by resource constraints. Focusing on aggregation queries, STREAM uses algorithms that determine at what points in a query plan should load shedding be performed and what

amount of load should be shed at each point in order to minimize the degree of inaccuracy introduced into query answers.

5 Reducing Memory Usage

Simply generating the straightforward query plans and executing them can be very inefficient in memory usage. STREAM uses the following techniques to reduce memory overhead: synopsis sharing to reduce data redundancy in synopsis; operator scheduling algorithm to reduce intermediate state; exploiting stream constraints to reduce run-time synopsis sizes.

The technique of exploiting constraints implemented in STREAM is k -constraints. It can also be utilized as the approximation technique when memory become insufficient, e.g., in the case of bursty data stream, or many complex queries with large windows. With k -constraints, memory usage can be reduced at the cost of query accuracy.

5.1 Synopsis Sharing

In the basic implementation of a query plan, multiple synopses within a single query plan may materialize nearly identical relations. This redundancy is eliminated by replacing the synopses with light-weight stubs, and a single store to hold the actual tuples. These stubs implement the same interfaces as non-shared synopses, so operators can be oblivious to the details of sharing.

5.2 Operator Scheduling

In a query plan each operator consumes elements from its input queues and produces elements on its output queue. The global operator scheduling policy can have a large effect on memory utilization, particularly with bursty input streams. STREAM uses a chain scheduling algorithm. The algorithm forms blocks “chains” of operators as follows: Start by marking the first operator in the plan as the “current” operator. Next, find the block of consecutive operators starting at the “current” operator that maximizes the reduction in total queue size per unit time. Mark the first operator following this block as the “current” operator and repeat the previous step until all operators have been assigned to chains. Chains are scheduled according to the greedy algorithm, but within a chain, execution proceeds in FIFO order. In terms

of overall memory usage, this strategy is provably close to the optimal “clairvoyant” scheduling strategy, i.e., the optimal strategy based on knowledge of future input.

5.3 Exploiting Constraints

Streams may exhibit certain data or arrival patterns that can be exploited to reduce run-time synopsis sizes. Such constraints can either be specified explicitly at stream-registration time, or inferred by gathering statistics over time.

STREAM identifies three types of useful constraints over data streams, referred to as three k -constraints:

- A referential integrity k -constraint on a many-one join between streams defines a bound k on the delay between the arrival of a tuple on the “many” stream and the arrival of its joining “one” tuple on the other stream.
- An ordered-arrival k -constraint on a stream attribute $S.A$ defines a bound k on the amount of reordering in values of $S.A$. Specifically, given any tuple s in stream S , for all tuples s' that arrive at least $k + 1$ elements after s , it must be true that $s'.A \geq s.A$.
- A clustered-arrival k -constraint on a stream attribute $S.A$ defines a bound k on the distance between any two elements that have the same value of $S.A$.

For each of the above k -constraints, query plan construction and execution algorithms are developed to take the stream constraints into account in order to reduce synopsis sizes at query operators by discarding unnecessary state. The smaller the value of k for each constraint, the more state that can be discarded.

Furthermore, if an assumed k -constraint is not satisfied by the data, the algorithm produces an approximate answer whose error is proportional to the degree of deviation of the data from the constraint. Therefore the k -constraints algorithm can also be used as an approximation where memory resources become insufficient.

6 Conclusion

In this report, we introduced STREAM, a Data Stream Management System. We focused on the Continuous Query Language supported by STREAM, the implementation of CQL, the techniques used in STREAM to reduce CPU and memory usage, and the inaccuracy in the query result incurred by these techniques.

References

- [1] The STREAM Group. Stanford Data Stream Management System. To appear in a book on data stream management edited by Garofalakis, Gehrke, and Rastogi.
- [2] A. Arasu, S. Babu and J. Widom. *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. To appear in VLDB Journal, 2005
- [3] B. Babcock, M. Datar, and R. Motwani. *Load Shedding for Aggregation Queries over Data Streams*. In Proc. of ICDE 2004, March 2004
- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. *Operator Scheduling in Data Stream Systems*. To appear in VLDB Journal, 2005
- [5] S. Babu, U. Srivastava, and J. Widom. *Exploiting k -Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams*. In ACM TODS, Sep. 2004