Query Processing, Resource Management, and Approximation in a Data Stream Management System*

Jennifer Widom
Stanford University

* Abridged by CZ

Data Streams

- Continuous, unbounded, rapid, time-varying streams of data elements
- Occur in a variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks
  - Telecom call records
  - Financial applications
  - Web logs and click-streams
  - Manufacturing processes
- DSMS = Data Stream Management System
DBMS versus DSMS

- Persistent relations
- Transient streams (and persistent relations)
DBMS versus DSMS

- Persistent relations
- One-time queries
- Transient streams (and persistent relations)
- Continuous queries
- Random access
- Sequential access
DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- Access plan determined by query processor and physical DB design

- Transient streams (and persistent relations)
- Continuous queries
- Sequential access
- Unpredictable data characteristics and arrival patterns

The STREAM System

- Data streams and stored relations
- Declarative language for registering continuous queries
- Flexible query plans
- Designed to cope with high data rates and query workloads
  - Graceful approximation when needed
  - Careful resource allocation and usage
- Relational, centralized (for now)
The (Simplified) Big Picture

**Using Conventional DBMS**

- Data streams as relation inserts, continuous queries as triggers or materialized views
- Problems with this approach
  - Inserts are typically batched, high overhead
  - Expressiveness: simple conditions (triggers), no built-in notion of sequence (views)
  - No notion of approximation, resource allocation
  - Current systems don’t scale to large # of triggers
  - Views don’t provide streamed results
- But we (and others) plan to compare
Declarative Language for Continuous Queries

• A distinction between STREAM and the Aurora project
  – Aurora users directly manipulate one large execution plan
  – STREAM compiles declarative queries into individual plans, system may merge plans
  – STREAM also supports direct entry of plans

• Syntax based on SQL, additional constructs for sliding windows and sampling

Example Query 1

Two streams, contrived for ease of examples:
Orders (orderId, customer, cost)
Fulfillments (orderId, clerk)
Example Query 1

Two streams, contrived for ease of examples:
Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk “Sue” for customer “Joe”

Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = “Sue”
And O.customer = “Joe”
Example Query 1

Two streams, contrived for ease of examples:
Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk “Sue” for customer “Joe”

Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = “Sue”
And O.customer = “Joe”
Example Query 1

Two streams, contrived for ease of examples:
Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk “Sue” for customer “Joe”

```
Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = “Sue”
   And O.customer = “Joe”
```

Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
   Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```
Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

Select F.clerk, Max(O.cost)
From Orders O,
   Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```sql
Select F.clerk, Max(O.cost)
From Orders O,
    Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

Semantics of Database Languages

- An often neglected topic
- Traditional relational databases are in reasonable shape
  - Relational algebra ? SQL
- But triggers were a mess
- The semantics of an innocent-looking continuous query over data streams may not be obvious
A Nonobvious Continuous Query

- Stream of stock quotes: Stocks(ticker,price)
- Monitor last 10 minutes of quotes:
  \[
  \text{Select } \_ \_ \_ \_ \_ \text{ From Stocks [Range 10 minutes]}
  \]
- Is result a relation, a stream, or something else?
- If a relation, what exactly does it contain?
- If a stream, how does query differ from:
  \[
  \text{Select } \_ \_ \_ \_ \_ \text{ From Stocks [Range 1 minute]}
  \]
  or \[
  \text{Select } \_ \_ \_ \_ \_ \text{ From Stocks [? ]}
  \]

Our Semantics and Language for Continuous Queries

- **Abstract**: interpretation for CQs based on certain “black boxes”
- **Concrete**: SQL-based instantiation for our system; includes syntactic shortcuts, defaults, equivalences
- **Goals**
  - CQs over multiple streams and relations
  - Exploit relational semantics to the extent possible
  - Easy queries should be easy to write, simple queries should do what you expect
Relations and Streams

- Assume global, discrete, ordered time domain (more on this later)
- Relation
  - Maps time $T$ to set-of-tuples $R$
- Stream
  - Set of $(tuple, timestamp)$ elements

Conversions

- Special operators: $Istream$, $Dstream$, $Rstream$
- Any relational query language
Conversion Definitions

• Stream-to-relation
  – $S[W]$ is a relation — at time $T$ it contains all tuples in window $W$ applied to stream $S$ up to $T$
  – When $W = T$, contains all tuples in stream $S$ up to $T$

• Relation-to-stream
  – $Istream(R)$ contains all $(r, T)$ where $r \in R$ at time $T$ but $r \notin R$ at time $T-1$
  – $Dstream(R)$ contains all $(r, T)$ where $r \notin R$ at time $T-1$ but $r \in R$ at time $T$
  – $Rstream(R)$ contains all $(r, T)$ where $r \in R$ at time $T$

Abstract Semantics

• Take any relational query language
• Can reference streams in place of relations
  – But must convert to relations using any window specification language
    (default window = $[?]$)
• Can convert relations to streams
  – For streamed results
  – For windows over relations
    (note: converts back to relation)
**Query Result at Time** $T$

- Use all relations at time $T$
- Use all streams up to $T$, converted to relations
- Compute relational result
- Convert result to streams if desired

**Time**

- Easiest: global system clock
  - Stream elements and relation updates timestamped on entry to system
- Application-defined time
  - Streams and relation updates contain application timestamps, may be out of order
  - Application generates “heartbeat”
    - Or deduce heartbeat from parameters: stream skew, scrambling, latency, and clock progress
  - Query results in application time
Abstract Semantics – Example 1

Select F.clerk, Max(O.cost)
From O, F [Rows 1000]
Where O.orderID = F.orderID
Group By F.clerk

• Maximum-cost order fulfilled by each clerk in last 1000 fulfillments

Abstract Semantics – Example 1

Select F.clerk, Max(O.cost)
From O, F [Rows 1000]
Where O.orderID = F.orderID
Group By F.clerk

• At time $T$: entire stream $O$ and last 1000 tuples of $F$ as relations
• Evaluate query, update result relation at $T$
Abstract Semantics – Example 1

Select `Istream(F.clerk, Max(O.cost))`
From O [? ], F [Rows 1000]
Where O.orderID = F.orderID
Group By F.clerk

- At time $T$: entire stream $O$ and last 1000 tuples of $F$ as relations
- Evaluate query, update result relation at $T$
- **Streamed result**: New element $(<\text{clerk}, \text{max}>, T)$ whenever $<\text{clerk}, \text{max}>$ changes from $T-1$

Abstract Semantics – Example 2

Relation `CurPrice(stock, price)`

Select stock, Avg(price)
From `Istream(CurPrice)` [Range 1 Day]
Group By stock

- Average price over last day for each stock
Abstract Semantics – Example 2

Relation CurPrice(stock, price)

Select stock, Avg(price)
From Istream(CurPrice) [Range 1 Day]
Group By stock

- Istream provides history of CurPrice
- Window on history, back to relation, group and aggregate

Concrete Language – CQL

- Relational query language: SQL
- Window spec. language derived from SQL-99
  - Tuple-based, time-based, partitioned
- Syntactic shortcuts and defaults
  - So easy queries are easy to write and simple queries do what you expect
- Equivalences
  - Basis for query-rewrite optimizations
  - Includes all relational equivalences, plus new stream-based ones
Two Extremely Simple CQL Examples

Select ? From Strm
  • Had better return Strm (It does)
    – Default ? window for Strm
    – Default Istream for result

Select ? From Strm, Rel Where Strm.A = Rel.B
  • Often want “NOW” window for Strm
  • But may not want as default

Query Execution

• When a continuous query is registered, generation a query plan
  – Users can also register plans directly

• Plans composed of three main components:
  – Operators (as in most conventional DBMS’s)
  – Inter-operator Queues (as in many conventional DBMS’s)
  – State (synopses)

• Global scheduler for plan execution
Memory Overhead in Query Processing

- Queues + State
- Continuous queries keep state indefinitely
- Online requirements suggest using memory rather than disk
  - But we realize this assumption is shaky
- **Goal: minimize memory use while providing timely, accurate answers**

Reducing Memory Overhead

Two main techniques to date

1) Exploit constraints on streams to reduce state
2) Clever operator scheduling to reduce queue sizes
http://www-db.stanford.edu/stream

Contributors: Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Justin Rosenstein, Rohit Varma