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

Jennifer Widom
Stanford University

Formula for a Database Research Project

• Pick a simple but fundamental assumption underlying traditional database systems
  – Drop it
• Reconsider all aspects of data management and query processing
  – Many Ph.D. theses
  – Prototype from scratch

Following the Formula

• We followed this formula once before
  – The LORE project
  – Dropped assumption:
    – Data has a fixed schema declared in advance
    – Semistructured data
• The STREAM Project
  – Dropped assumption:
    – Data lives as persistent data sets
    – Data streams

Data Streams

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

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
• Designed to cope with high data rates and query workloads
  – Graceful approximation when needed
  – Careful resource allocation and usage
• Relational, centralized (for now)
Concrete Contributions to Date

- Semantics for continuous queries
- Query plans
- Exploiting stream constraints
- Approximation techniques
- Resource allocation to maximize precision
- Operator scheduling
- Initial prototype

The (Simplified) Big Picture

Concise representation of data flow:
- Input streams
- Register
- Query
- Streamed Result
- Stored Result
- Archive
- Scratch Store
- Stored Relations
- Register Monitoring Queries
- Online Performance Metrics
- DSMS
- Network measurements, Packet traces
- Scratch Store
- Lookup Tables
- Archive

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 plan to compare

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 plan to compare

Declarative Language for Continuous Queries

- A distinction between STREAM and Aurora
  - 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)

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 Preceding] 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 Preceding]
10% Sample
Where O.orderID = F.orderID
Group By F.clerk

Query Language Semantics
• I had a revelation around 1989
  People were inventing and implementing trigger (active rule) systems without completely defining or understanding rule behavior
• I had another one around 2000
  Ditto for continuous queries
• And yet another one just a month ago…

Last Month’s Revelation
• The semantics of SQL queries is (relatively) easy to understand
  – Even lots of SQL queries running together
• The semantics of a single trigger is (relatively) easy to understand
  – But lots of triggers together can be complex
• The semantics of even a single continuous query may not be obvious
  – But lots running together is no harder

Our Previous Working Semantics
“The result of a continuous query at time T is the result of treating the streams up to T as relations and evaluating the query using standard relational semantics.”
+ Relies on existing well-understood semantics
  – Asymmetric: streams-to-relations but not relations-to-streams
  – Questionable for complex queries (subqueries, aggregation)
  – Assumes global time

Our Elaborated Semantics
• Comes in two parts
  – Abstract: window specification language and relational query language as “black boxes”
  – Concrete: SQL-based instantiation for our system; includes syntactic shortcuts, defaults, equivalences
• Based on two concepts
  – Relation
  – Stream

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
  – Stream “at [up to] time T” = all elements with timestamp ≤ T
Conversions

- **Stream-to-relation**
  - \(relation(S, W)\) at time \(T\) contains all tuples in window \(W\) applied to stream \(S\) at time \(T\)
  - When \(W = \infty\), all tuples in stream \(S\) with timestamps \(\leq T\)

- **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 \(r \in R\) at time \(T-1\) but \(r \notin R\) 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 (including default \(W = \infty\))
- 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 \(Istream\) (and/or \(Dstream\)) if desired

Abstract Semantics – Example 1

Select \(F\.clerk\), \(Max(O\.cost)\)
From \(O\) \([-\infty\], \(F\) [Rows 1000 Preceding]
Where \(O\.orderID = F\.orderID\)
Group By \(F\.clerk\)
- Maximum-cost order fulfilled by each clerk in last 1000 fulfillments
- Stored or streamed result

Time

- Relevant for relation state, stream timestamps, query results
- Easiest: global system clock
- But application-defined “time” works too
- Aside: temporal and sequence databases
  - One-time queries, stored results
  - Some features overkill for data stream applications (we think)

Abstract Semantics – Example 1

Select \(F\.clerk\), \(Max(O\.cost)\)
From \(O\) \([-\infty\], \(F\) [Rows 1000 Preceding]
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 \(T\)
- \(Istream(query)\): New element \(<clerk, max>, T\) whenever \(<clerk, max>\) changes from \(T-1\)
Abstract Semantics – Example 2
Relation CurPrice(stock, price)
Select stock, Avg(price)
From Istream(CurPrice) [Range 1 Day Preceding]
Group By stock
• Average price over last day for each stock

Concrete Language – CQL
• Relational query language: SQL
• Window specifications
  – Tuple-based, time-based, PARTITION
• Syntactic shortcuts and defaults
  – So easy queries are easy to write
  – So queries do what you expect
• Equivalences
  – Basis for optimizations

Semantics – Not Covered
• Sampling
  – Obvious semantics works fine
• Answer production, especially with application-defined time, out-of-order streams
  – Formalized, relatively straightforward
• Implementing this semantics vs. something less strict
  – Under consideration

Abstract Semantics – Example 2
Relation CurPrice(stock, price)
Select stock, Avg(price)
From Istream(CurPrice) [Range 1 Day Preceding]
Group By stock
• Istream provides history of CurPrice
• Window on history, back to relation, group and aggregate

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 Plans
• No interesting query plan generation yet
  – Queries translated directly to “naïve” plan
• Plans composed of three fundamental components:
  – Operators (as in most conventional DBMS’s)
  – Inter-operator Queues (as in many conventional DBMS’s)
  – Synopses
• Global scheduler for plan execution
Synopses and Operators

- **Synopses**
  - Summarize tuples seen so far (exact, approximate)
  - For sliding windows
- **Currently synopses are tied to operators**
  - Generic interfaces allow any synopsis type with any operator type
  - Will eventually decouple (e.g., for sharing)
- **Example: synopsis join**

Example Query Plan

- **Synopses**
  - Unbounded for many queries [PODS '02]
  - Can reduce size using stream constraints
  - But eventually may need to resort to approximate results

Memory Overhead

- **Plan state = synopses + queues**
- 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**

Memory Overhead (cont’d)

- **Queues**
  - Scheduling algorithm with near-optimal performance in maximum queue size
- **Synopses**
  - Unbounded for many queries [PODS ‘02]
  - Can reduce size using stream constraints
  - But eventually may need to resort to approximate results

Exploiting Stream Constraints

- **Unbounded (or very large) synopses often required for “arbitrary” streams**
- But streams may exhibit constraints that reduce, bound, or even eliminate synopses
- **Stream constraints can be…**
  - Declared as part of CREATE STREAM
  - Discovered over time (through statistics)
  - Declared dynamically in data – punctuations [Tucker, Maier, et al.]

Constraints We Consider

- **Four constraint types:**
  - Many-one join (assumed)
  - Referential integrity for join
  - Clustered arrival for attribute
  - Ordered arrival for attribute
- **Relaxed versions:** adherence parameter $k$
  - Ref-Integrity($k$) for $S_1 \rightarrow S_2$
  - Clustered($k$) for S.A
  - Ordered($k$) for S.A
Algorithm for Exploiting Constraints

- **Input**
  - Any Select-Project-Join query over streams
  - Any combination of $k$-constraints
- **Output**
  - Query execution plan that reduces or eliminates synopses based on $k$-constraints
  - If constraints violated, get approximate result

Constraint Examples

Orders (orderID, customer, cost)
Fulfillments (orderID, portion, clerk)
Query: Many-one join $F \rightarrow O$

- **Ref-integrity($k$)**: $F$ tuples "wait" in their synopsis for at most $k$ arrivals of $O$ tuples
- **Clustered($k$) on $F$.orderID**: Matched $O$ tuples deleted from synopsis after $k$ arrivals of non-matching $F$'s

Note: adherence parameter $k$ is tuple-based, could be time-based instead

Approximation

Why approximate?
- Memory requirement too high, even with constraints
- Can’t process streams fast enough for query load

Approximation (cont’d)

- Static: rewrite queries to add (or shrink) sampling or windows
  - User can participate, predictable behavior
  - Doesn’t consider dynamic conditions
- Dynamic: modify query plan – insert sampling operators, bounded-size synopses, load shedding
  - Adapts to current resource availability
  - How to convey to user? (major open issue)

Resource Allocation to Maximize Precision

- Static or dynamic, based on query plan
- Any resource (think memory)
- Assumptions
  - Query plans are trees of operators
  - Each operator has function from resource allocation $R$ to precision $P$
  - Simple $(FP, FN)$ precision model
    - $FP =$ probability of answer being false positive
    - $FN =$ # false negatives per correct answer

Precision of an Operator

- $FP =$ False Positives
- $FN =$ False Negatives
Precision of Two Operators

The General Problem

- Given a plan, precision function for each operator in the plan, and \( R \) total resources, allocate \( R \) to operators to maximize result precision
- Solution
  - For each operator type: formula for calculating output precision given input precision and operator resource allocation
  - Assume <0,0> precision for input streams
  - Becomes nonlinear optimization problem

The Holy Grail

- Given:
  - Declarative query
  - Resources
  - Constraints on streams
  - Generate plan and resource allocation that takes advantage of constraints and maximizes precision
- Do it for multiple (weighted) queries, dynamically and adaptively, and convey what’s happening to the user

Conclusion

- We have some interesting pieces
- We’re putting them together in a prototype
  
  http://www-db.stanford.edu/stream

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