Data Streams & Continuous Queries

The Stanford STREAM Project

stanfordstreamdatamanager
Data Streams

• Continuous streams of data elements (may be unbounded, rapid, time-varying)

• Occur in a variety of modern applications
  – Network monitoring and traffic engineering
  – Sensor networks, RFID tags
  – Telecom call records
  – Financial applications
  – Web logs and click-streams
  – 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 (Simplified) Big Picture

Input streams → DSMS → Scratch Store → Stored Relations → Archive

Register Query → DSMS → Streamed Result

Stored Result
(Simplified) Network Monitoring

Register Monitoring Queries

Intrusion Warnings

Online Performance Metrics

Network measurements, Packet traces

DSMS

Scratch Store

Lookup Tables

Archive

Stanfordstreamdatamanager
The **STREAM System**

- Data streams and stored relations
- SQL-based language for registering continuous queries
- Variety of query execution strategies
- Textual, graphical, and application interfaces
- Relational, centralized
Rest of This Lecture

- Query language
- System issues and overview (brief)
- Live system demonstration
Goals in Language Design

1) Support continuous queries over multiple streams and updateable relations
2) Exploit existing relational semantics to the extent possible
3) Easy queries should be easy to write
4) Simple queries should do what you expect
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
Next

- Formal definitions for relations and streams
- Formal conversions between them
- Abstract semantics
- Concrete language: CQL
- Syntactic defaults and shortcuts
- Equivalence-based transformations
Relations and Streams

• Assume global, discrete, ordered time domain

• Relation
  – Maps time $T$ to set-of-tuples $R$

• Stream
  – Set of $(tuple, timestamp)$ elements
Conversions

Window specification

Streams

Special operators: $Istream$, $Dstream$, $Rstream$

Relations

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 $W = \infty$, 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 $r \in R$ at time $T-1$ but $r \notin R$ 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 = $[\infty]$)

• 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 streams if desired
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 \( \text{Istream}(F.clerk, \text{Max}(O.cost)) \)
From \( O [\infty], F [\text{Rows 1000}] \)
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 \( T-1 \)
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 specification language derived from SQL-99
  - Tuple-based windows
  - Time-based windows
  - Partitioned windows
- Simple “X% Sample” construct
CQL (cont’d)

• 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

• Examples: already seen some, more coming up
Shortcuts and Defaults

• Prevalent stream-relation conversions can make some queries cumbersome
  – *Easy queries should be easy to write*

• Two defaults:
  – **Omitted window**: Default \([\infty]\)
  – **Omitted relation-to-stream operator**: Default *Istream* operator on:
    • Monotonic outermost queries
    • Monotonic subqueries with windows
The Simplest CQL Query

Select * From Strm

• Had better return Strm (It does)
  – Default \([\infty]\) window for Strm
  – Default Istream for result
Simple Join Query

Select * From Strm, Rel Where Strm.A = Rel.B

- Default $[\infty]$ window on *Strm*, but often want *Now* window for stream-relation joins

Select Istream(O.orderID, A.City)
From Orders O, AddressRel A
Where O.custID = A.custID
Simple Join Query

Select * From Strm, Rel Where Strm.A = Rel.B

- Default [$\infty$] window on Strm, but often want Now window for stream-relation joins

Select Istream(O.orderID, A.City)
From Orders O [$\infty$], AddressRel A
Where O.custID = A.custID
Simple Join Query

Select * From Strm, Rel Where Strm.A = Rel.B

• Default $[\infty]$ window on Strm, but often want Now window for stream-relation joins

Select Istream(O.orderID, A.City)
From Orders O $[\text{Now}]$, AddressRel A
Where O.custID = A.custID

• We decided against a separate default
Equivalences and Transformations

• All relational equivalences apply to all relational constructs directly
  – Queries are highly relational

• Two new transformations
  – Window reduction
  – Filter-window commutativity
Window Reduction

Select Istream(L) From S \([\infty]\) Where C is equivalent to

Select Rstream(L) from S [Now] Where C

• Question for class
  ➢ Why \(Rstream\) and not \(Istream\) in second query?

• Answer: Consider stream \(<5>, <5>, <5>, <5>, \ldots\)
Window Reduction (cont’d)

Select Istream(L) From S [∞] Where C is equivalent to

Select Rstream(L) from S [Now] Where C

- First query form is very common due to defaults
- In a naïve implementation second form is much more efficient
Another question for class

When is

\[
\text{Select L From S [window] Where C}
\]
equivalent to

\[
\text{Select L From (Select L From S Where C) [window]}
\]

Is this transformation always advantageous?
Constraint-Based Transformations

• Recall first example query (simplified)
  Select  Sum(O.cost)
  From Orders O, Fulfillments F [Range 1 Day]
  Where O.orderID = F.orderID

• If orders always fulfilled within one week
  Select  Sum(O.cost)
  From Orders O [Range 8 Days],
   Fulfillments F [Range 1 Day]
  Where O.orderID = F.orderID

• Useful constraints: keys, stream referential integrity, clustering, ordering
STREAM System

• First challenge: basic functionality from scratch

• Next steps – cope with:
  – Stream rates that may be high, variable, bursty
  – Stream data that may be unpredictable, variable
  – Continuous query loads that may be high, variable

➢ Overload

➢ Changing conditions
System Features

• Aggressive **sharing** of state and computation among registered queries

• Careful **resource allocation and use**

• Continuous **self-monitoring and reoptimization**

• Graceful **approximation** as necessary
Query Execution

• When a continuous query is registered, generate a query plan
  – New plan merged with existing plans
  – Users can also create & manipulate plans directly

• Plans composed of three main components:
  – Operators
  – Queues (input and inter-operator)
  – State (windows, operators requiring history)

• Global scheduler for plan execution
Simple Query Plan
System Status

• System is “complete”
  – 30,000 lines of C++ and Java
  – Multiple Ph.D. theses, undergrad and MS projects

• Source is available and system is being used

• Can also use system over internet
Stream System Benchmark: “Linear Road”

10 Expressways

100 segments of 1 mile each

Linear City

Reports every 30 seconds

Main Input Stream: Car Locations (CarLocStr)

<table>
<thead>
<tr>
<th>car_id</th>
<th>speed</th>
<th>exp_way</th>
<th>lane</th>
<th>x_pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>55</td>
<td>5</td>
<td>3 (Right)</td>
<td>12762</td>
</tr>
<tr>
<td>1035</td>
<td>30</td>
<td>1</td>
<td>0 (Ramp)</td>
<td>4539</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
STREAM System Demo

- Incoming data streams
- Continuous queries executing over streams
- Query plan visualizer
- System monitoring via “introspection” queries
- Benchmark execution