Streaming Queries over Streaming Data

Lukasz Golab
CS856
University of Waterloo

Table of Contents

- The Data Stream Model
- Continuous (Streaming) Queries
- Streaming Queries over Streaming Data
  - State Modules (SteMs)
  - Eddies (adaptive query processing)
  - PSoup

Goals of this Talk

- Data Stream Model for on-line information sources
- Querying data streams via continuous queries
- PSoup solves the problem of continuous queries over data streams

The Data Stream Model

- Definition: real-time, continuous sequence of items
- Applications: on-line financial tickers, sensor networks, Internet traffic monitoring
- Properties: real-time, high arrival rate, infinite length, ordering
- High-level view: stream of relational tuples

Why the Data Stream Model?

- Data access: push, not pull (consider pervasive computing)
- Scale: collection of streaming data sources vs. relational tables/XML
- Information freshness: data on the Web is changing, new data replace old data

Continuous (Streaming) Queries

- Run persistently over a period of time
- Return new results as new data items arrive
- Predefined or ad-hoc
- Landmark or sliding window

Why Continuous Queries?

- Fit the data stream model
- "What is the average salary in the Toy department?" vs. "What was the average temperature in room x over the past 24 hours?"
- Note: sliding window queries over recent history likely to be most popular

Streaming Queries over Streaming Data

- System Requirements
  - Scalability
  - Adaptivity (e.g. disconnected operation)
  - Performance
  - Memory Constraints
    - With finite memory, can do $\sigma_{R.a<5}(R)$
    - But can't do $\sigma_{R.a=S.b}(R \times S)$

Streaming Queries over Streaming Data (2)

- Solution
  - State Modules (SteMs)
  - Eddies (adaptive query processing)
  - PSoup

Background—SteMs

- Used for interactive query processing, e.g. data sharing among joins
- One SteM for each relation
- Can insert, delete, index etc like a relational table + can probe
- E.g. $\pi_{R.a,\ S.b}\sigma_{R.a<5}(R \bowtie_{R.a>S.b} S)$

SteMs example

R.a		S.b
2	-> $\pi_{R.a,\ S.b}(R \bowtie_{2>S.b} S)$	2
3	-> $\pi_{R.a,\ S.b}(R \bowtie_{3>S.b} S)$	3
4	-> $\pi_{R.a,\ S.b}(R \bowtie_{4>S.b} S)$	4
9		5
7		0

Background—Eddies

- Execute operators in different order throughout the lifetime of the query
- Choose a plan that is cheapest at any given time
- Tuple routing policies
- e.g. $(R \bowtie S) \bowtie T$ vs. $R \bowtie (S \bowtie T)$

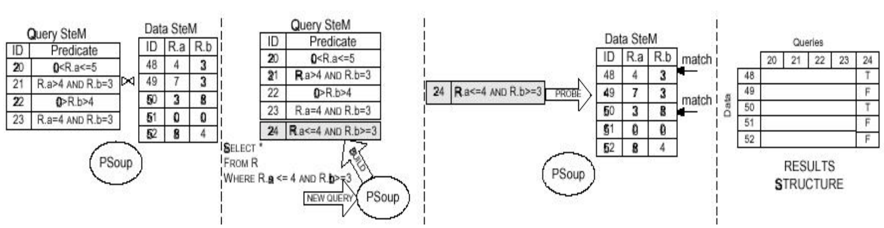
Why adaptivity?

- 3 things influence the cost of a plan
  - Changing input (stream) rates
  - Changing operator processing times (e.g. memory/resource sharing)
  - Changing selectivities—e.g.
    - FACULTY table with a clustered index on AGE
    - Want SELECT NAME FROM FACULTY
      WHERE SALARY>100000
    - **Selectivity changes from 0 to 1!**

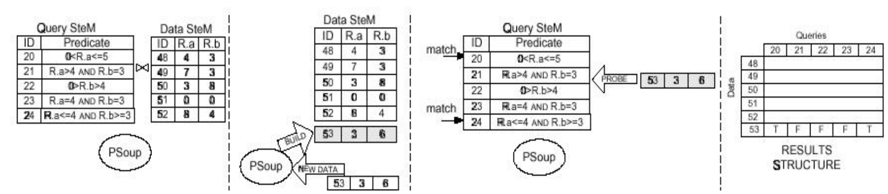
PSoup

- Insight: treat queries like tuples
- Implementation:
  - One Query SteM, many Data SteMs
  - Results Structure
  - Eddies—adaptive ordering of data-query joins

Processing a New Query



Processing New Data



PSoup—Notes

- Results structure and indices stored in main memory
- Supports sliding window queries (tuples have timestamps)
- Join processing is more complicated (results structure is bigger)
- Supports disconnected operation—user can get the data at any time

Summary

- PSoup meets all system requirements:
  - Scalability—probing the Query SteM essentially executes all queries at once
  - Adaptivity—Eddies + Results Structure. Note that new queries may access old data
  - Performance—indices on query predicates, tables (in this case stream excerpts) and columns in the Results Structure
  - Memory Constraints—Instead of trying to do arbitrary queries, focus on windowed joins