Streaming Queries over Streaming Data

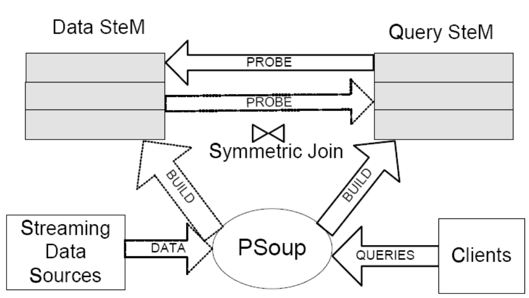
Sirish Chandrasekaran and Michael J. Franklin
University of California at Berkeley
VLDB 2002

Presented by:
George Beskales

Motivation and Contribution

- Current Systems support either
  - Streaming Queries over static data (traditional DBMS)
  - Static queries over streaming data (Data Streaming Systems)
- PSoup supports streaming queries streaming data.
  - Data Streams and Query Stream
  - New queries can access old data (and of course new data)
  - Active / Inactive queries (,i.e. disconnected operation)
  - Query results is partially materialized

PSoup System Architecture



Query Structure

```
SELECT select_list
FROM from_list
WHERE conjoined_boolean_factors
BEGIN begin_time
END end_time
```

Standing Query Clause (SQC): SELECT, FROM, WHERE

Begin_End clause: BEGIN, END

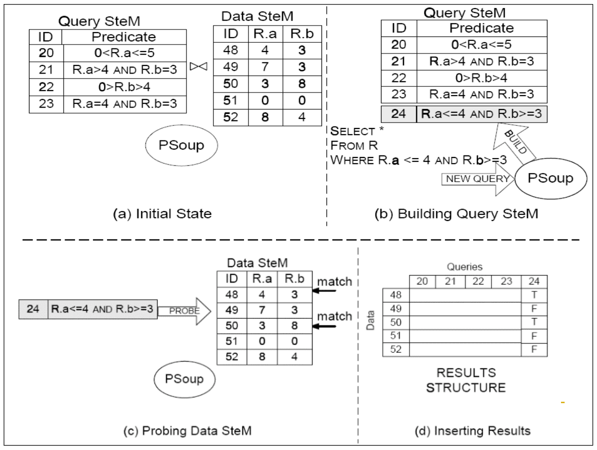
Modes of Query

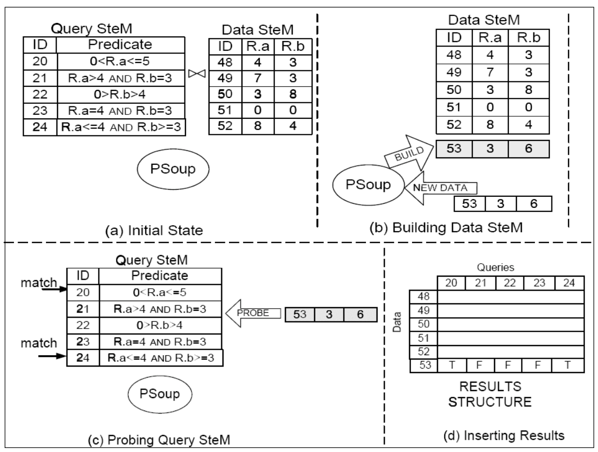
- Snapshot : begin_time & end_time are constants
- Landmark : begin_time is constant, end_time is variable (e.g. NOW)
- Sliding Window :begin_time & end_time are variables.

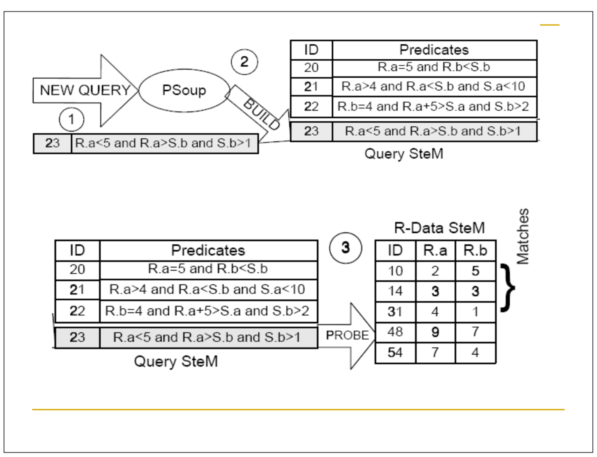
- PSoup assumes that sliding window technique is used and it fits into the main memory.

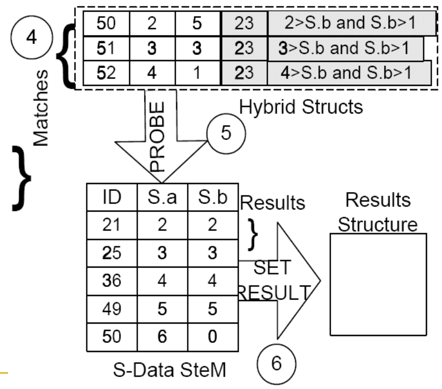
Data structures

- Data State Module (SteM): holds the current tuples for each data source.
- Query State Module (SteM): stores SQCs of all queries.
- WindowTable : stores Begin_End clause of the queries
- Results Structure : Holds (partially) materialized results
- Hybrid Struct : to hold intermediate join results.


Query SteM
Data SteM
PSoup
(a) Initial State
SELECT *
FROM R
WHERE R.a <= 4 AND R.b>=3
NEW QUERY
BUILD
(b) Building Query SteM
24 R.a<=4 AND R.b>=3
PROBE
match
(c) Probing Data SteM
Queries
Data
RESULTS
STRUCTURE
(d) Inserting Results


Query SteM
Data SteM
PSoup
(a) Initial State
BUILD
NEW DATA
(b) Building Data SteM
match
PROBE
(c) Probing Query SteM
Queries
Data
RESULTS
STRUCTURE
(d) Inserting Results


NEW QUERY
PSoup
BUILD
23 R.a<5 and R.a>S.b and S.b>1
Query SteM
R-Data SteM
PROBE
Matches



Implementation Issues

- Eddy is modified to be *Stream-Prefix-Consistent*
  - Temporary tuples are stored separately from new tuples.
  - Temporary tuples are processed before new tuples.
- Data SteM
  - Red-Black tree indexes are created for every attribute of each stream
  - Hash index over tupleID to speed up result construction

Implementation Issues

- Query SteM
  - Red-Black index over predicates constants, e.g. *c* in predicate (R.a > c)
  - Each node has five lists, one for each RELOP <,≤ ,= ,≥ ,>
  - Predicates that have more than one attribute are stored in linked list.
  - AND operators are implemented by decrement of a counter until it reaches zero

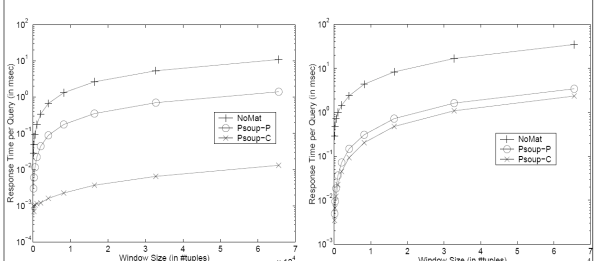
Implementation Issues

- Results Structure
  - Each cell refer to a query and a tuple
  - 2D bitmap (tuple timestamp, query ID)
  - Linked list for each query
  - Timestamp in case of streams joins is the older based on assumption that Snapshot queries are less frequent.

Experiments

- Psoup-P : lazy approach; results are output when requested (partial materialization)
- Psoup-C : eager approach; results are output immediately (complete materialization)
- NoMat : does not materialize results

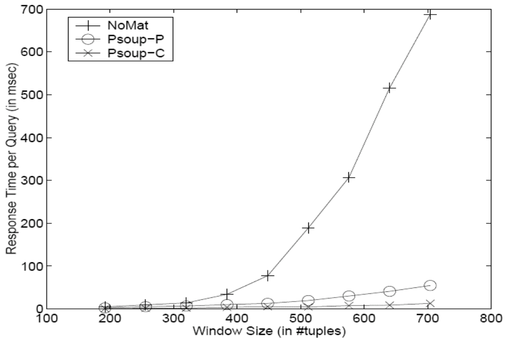
Response time vs. window size



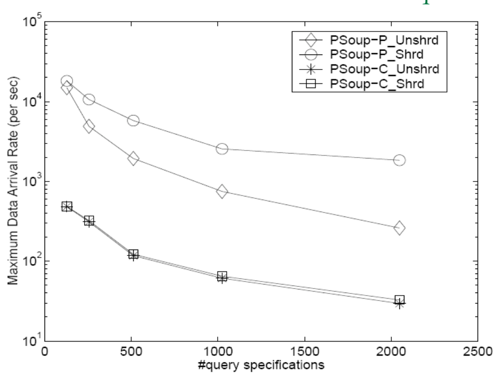
(a) Equality Predicates

(b) Interval Predicates

Response time vs. window size (Joins Queries)

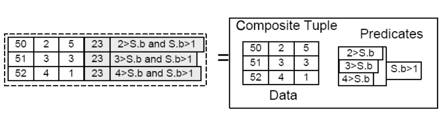


Max data arrival rate vs. number o queries

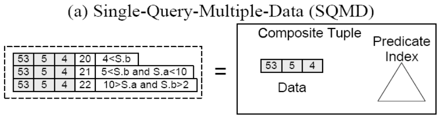


Extensions to PSoup

- Composite tuples in joins :



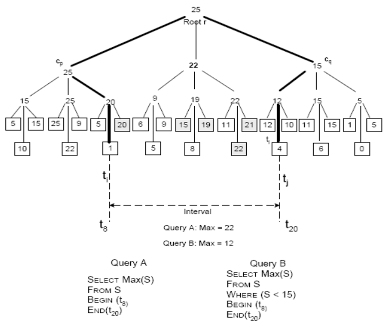
(a) Single-Query-Multiple-Data (SQMD)



(b) Single-Data-Multiple-Query (SDMQ)

Extensions to PSoup

- Aggregate queries



Pros

- Provide access to old data for new queries.
- Combination of efficient data processing rate and query response time by *partial* materialization and indexing data streams and query predicates.
- Support disconnection mode to avoid unnecessary maintaining of sliding window.

Cons

- Predicate Indexing is inefficient for complex predicates , e.g. string predicates, and complex mathematical predicates
- Index maintaining / materialization can be a bottleneck for high speed streams
- Sliding windows must completely reside in memory.
- How snapshot / landmark queries are processed.
- Maximum sustainable rate of queries and rate of invocations should be examined.
- Aggregate function are supported on small scale
- Query operator Scheduling is ignored
- Memory requirements are expected to be high.

Discussions

- How to support complex predicates without sacrificing the performance?
- How to integrate more sophisticated scheduling techniques
- What is the expected performance relative to other (newer) approaches (e.g. Aurora ad-hoc queries)
- What is the PSoup-P performance at different invocation rates / query rates
- Lazier approach that PSoup-P, especially if invocation rate is very low, e.g. selectively choose what attribute/query to materialize.
- How memory usage behave with different values of window size/ data rates.