PSoup
Streaming Queries of Streaming Data

Jonathan Abbett
The Plan

• Motivations
• Definition
• Queries
• Data
• Implementation
• Example
Motivations
Shortcomings of current DB systems

- **Traditional DBMSs**
  - Queries “streamed,” data static
  - Results only for distinct moment in time

- **Continuous Query Systems**
  - Queries static, data streamed
  - Results given continuously

- **Requirements of an ideal solution**
  - Support for streaming data and changing queries
  - Maintain some data over a period of time
  - Offer periodic synch or data summary
What is PSoup?
A preliminary overview

• Constantly changing queries are “registered” with the system
• Queries support “BEGIN-END” clause for windowing
• State Modules (SteMs)
  • Queries and data cached into SteMs for short-term storage
• Query-Data Joins
  • “Multiquery processing: a join of query and data streams.”
  • Query SteMs joined to Data SteMs to produce result
• Operates in main memory
PSoup Visualized

Query SteM
Stored queries

Data SteM
Cached data for a single stream

Joins between queries and required data
In practice

New query on one data stream

Initial state of our data

<table>
<thead>
<tr>
<th>ID</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Loc = Volen</td>
</tr>
<tr>
<td>11</td>
<td>Heartrate &gt; 130</td>
</tr>
<tr>
<td>12</td>
<td>Loc != Gosman AND Heartrate &gt; 130</td>
</tr>
</tbody>
</table>

Data SteM

<table>
<thead>
<tr>
<th>ID</th>
<th>Loc</th>
<th>HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Volen</td>
<td>150</td>
</tr>
<tr>
<td>22</td>
<td>Gzang</td>
<td>50</td>
</tr>
<tr>
<td>23</td>
<td>Gosman</td>
<td>180</td>
</tr>
<tr>
<td>24</td>
<td>Gosman</td>
<td>105</td>
</tr>
</tbody>
</table>

Query SteM

New query added

<table>
<thead>
<tr>
<th>ID</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Loc = Volen</td>
</tr>
<tr>
<td>11</td>
<td>Heartrate &gt; 130</td>
</tr>
<tr>
<td>12</td>
<td>Loc != Gosman AND Heartrate &gt; 130</td>
</tr>
<tr>
<td>13</td>
<td>Loc = Gzang AND Heartrate &lt; 60</td>
</tr>
</tbody>
</table>

One match in the stream

<table>
<thead>
<tr>
<th>ID</th>
<th>Loc</th>
<th>HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Gzang</td>
<td>50</td>
</tr>
</tbody>
</table>

Results bitmap updated
• Support for SELECT-FROM-WHERE queries
  • SELECT * FROM DataStream ds WHERE (ds.value1 < 5)
• Additional support for BEGIN-END clause
  • SELECT * FROM DataStream ds WHERE (ds.value1 < 5)
    BEGIN (NOW - 10)
    END (NOW)
  • In this example, query will return all qualifying results between ten minutes ago and now, based on server time.
Three styles.

• **Snapshot:** Constant BEGIN and END times
  - Data between class start and end...
    - START (1:30PM) END (4:30PM)

• **Landmark:** Constant BEGIN, variable END
  - Data from class start to present
    - START (1:30PM) END (NOW)

• **Sliding Window:** Variable BEGIN, variable END
  - Data from the last 45 minutes
    - START (NOW - 45) END (NOW)
New Query Handling

What happens when a query is entered.

- User registers query with Psoup and receives a unique QueryID
  - Can invoke QueryID at another time
  - New results “materialized” into Results Structure
- Query split into two parts
  - Standing Query Clause/BEGIN-END Clause
- SQC used for initial probe of data
  - Puts results from current data store into Results Structure
New Data Handling

What happens when new data arrives.

- Assigned a tupleID and timestamp
- Data is inserted into Data SteM
- Data tuple used for probe of standing queries
  - Adds itself to Results Structure of appropriate queries
State Modules
... or SteMs ...

- **Abstract data structures**
  - Used to store data and queries
  - Indexed for access speed

- **Data SteMs**
  - Stores, indexes stream data
  - One SteM for each stream
  - One tree index for each item

- **Query SteM**
  - Stores, indexes predicates of standing queries
  - One for entire system
Results Structure

Stores available results for all queries

- Indicates which tuples match with SQCs
- Data dropped when expired
- Bitmap and linked-list implementations
- Tuples sorted by timestamp
  - Speeds up application of window
In practice
New datum on one data stream.

Initial state of our data

<table>
<thead>
<tr>
<th>ID</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Loc = Volen</td>
</tr>
<tr>
<td>11</td>
<td>Heartrate &gt; 130</td>
</tr>
<tr>
<td>12</td>
<td>Loc != Gosman AND Heartrate &gt; 130</td>
</tr>
<tr>
<td>13</td>
<td>Loc = Gzang AND Heartrate &lt; 60</td>
</tr>
</tbody>
</table>

Query SteM

Matches one existing query

<table>
<thead>
<tr>
<th>ID</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Loc = Volen</td>
</tr>
<tr>
<td>11</td>
<td>Heartrate &gt; 130</td>
</tr>
<tr>
<td>12</td>
<td>Loc != Gosman AND Heartrate &gt; 130</td>
</tr>
<tr>
<td>13</td>
<td>Loc = Gzang AND Heartrate &lt; 60</td>
</tr>
</tbody>
</table>

Data SteM

New data item added

<table>
<thead>
<tr>
<th>ID</th>
<th>Loc</th>
<th>HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Volen</td>
<td>150</td>
</tr>
<tr>
<td>22</td>
<td>Gzang</td>
<td>50</td>
</tr>
<tr>
<td>23</td>
<td>Gosman</td>
<td>180</td>
</tr>
<tr>
<td>24</td>
<td>Gosman</td>
<td>105</td>
</tr>
<tr>
<td>25</td>
<td>Shapiro</td>
<td>175</td>
</tr>
</tbody>
</table>

Results bitmap updated

Probes
In practice
New query on two data streams.

New query added

Hybrid structure for matches in R

Update results structure
What to take away

• Multiple queries on multiple data streams
• Data cached within limits of memory
  • Allows summary results
  • Allows disconnect (i.e. data recharging)
• Queries “joined” to Data Streams to produce results
• Windows handily supported with BEGIN-END