Approximate Join Processing
Over Data Streams

Abhinandan Das      Johannes Gehrke
Mirek Riedewald
Cornell University

SAKIRE ARSLAN
USC - 2003
Outline

• Data Stream Join Processing
• Sliding Window Join
• Approximate Join
• Error Measures
• Join Algorithms using the Proposed Error Measure
  – Static algorithm
  – Offline algorithm with Fast CPU
  – Online algorithm with Fast CPU
• Experiments and Results
Data Stream Join Processing

- The data elements in the stream arrive online.
- The system has no control over the order in which data elements arrive to be processed.
- Once an element from a data stream has been processed it is discarded or archived.
- Data streams are potentially unbounded in size.
- Performing join operation on unbounded streams has high resource requirements (both CPU and memory).
Sliding Window Join

- Restrict the set of tuples that participate in the join to a bounded size window
- Window boundaries can be defined based on:
  - Time units
  - Number of tuples
  - Landmarks
- In proposed model: The window is defined in terms of time units, and at each time unit a new tuple arrives

window size = w

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>4</th>
<th>3</th>
<th>19</th>
<th>9</th>
<th>14</th>
<th>0</th>
<th>2</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>t-w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>t</td>
</tr>
</tbody>
</table>
Sliding Window Join (cont.)

- A sliding window join of window size $w$:
  - Has to store $2w$ tuples
  - Has to process incoming tuples as fast as they arrive

- **Problem**: Limited resources (storage and CPU)
- **Solution**: Approximating the output
Approximating Query Answers

• Load Shedding: Dropping tuples before they naturally expire
  – Drop the tuples randomly
  – Assign priorities to tuples and remove the lowest priority

• Proposed Solution: Semantic Load Shedding
  Which tuples should be dropped when—in order to minimize the error of the output
Join Processing Models

- Modular vs. Integrated
Join Processing Models (cont.)

• If CPU is fast:
  – Incoming tuples can be processed at least as quickly as they arrive
  – Modular and integrated models are equivalent
  – Approximation is due to memory restriction
  – Optimization Goal: Decide which tuples to drop in the join memory so that approximation error is minimized

• If CPU is slow:
  – Tuples arrive faster than they can be processed
  – Approximation is due to both memory and CPU processing constraints.
  – Optimization Goal: Select the tuples to drop in the join memory and the queue so that approximation error is minimized
Error Measures to Evaluate Approximation

• The output of the join operation is set a of tuples.
• For sets X & Y:
  – Symmetric Difference Measure is defined as
    \[| (X-Y) \cup (Y-X) |\]
• Proposed Error Measure: MAX-subset measure
  – MAX-subset measure represents the number of missing tuples in the approximate result set
  – It is a special case of Symmetric Difference Measure where one of the sets is a subset of the other
Error Measures to Evaluate Approximation (cont.)

• **MAX-subset measure**

\[ X = \text{the approximate result set} \]
\[ Y = \text{the exact result set} \]
\[ X \subseteq Y \]

symmetric difference \((X, Y) = |Y - X|\)

MAX-subset measure \((X, Y) = |Y - X|\)

• If the set \(X\) maximized the error will be minimized (similarly similarity will be maximized)
Error Measures to Evaluate Approximation (cont.)

Some of the set-theoretic error/similarity measures are:

1. Matching Coefficient: $|X \cap Y|$
2. Dice Coefficient: $2 \times |X \cap Y| / (|X| + |Y|)$
3. Jaccard Coefficient: $|X \cap Y| / |X \cup Y|$
4. Cosine Coefficient: $|X \cap Y| / |X \cup Y|^{1/2}$
5. Earth Mover’s Distance
6. Match and Compare
Join Algorithms

• Algorithm for the Static Case
• Offline window join algorithm with a Fast CPU
• Online window join algorithm with a Fast CPU
Bipartite Graphs

• A **bipartite graph** is a graph $G$ whose vertex set $V$ can be partitioned into two non empty sets $V_1$ and $V_2$ in such a way that every edge of $G$ joins a vertex in $V_1$ to a vertex in $V_2$.

  
  \[
  V_1 = \{1,4,6,7\} \\
  V_2 = \{2,3,5,8\}
  \]

• **Kuratowski's theorem**: a graph is **planar** if and only if it does not contain a subgraph which is an expansion of $K_5$ (the full graph on 5 vertices) or $K_{3,3}$ (six vertices, three of which connect to each of the other three)

• **Kuratowski components** are the graphs that follow Kuratowski's theorem
Static Case

- Input relations (A and B) are not data streams
- Goal is to find a set of k tuples to be dropped from the input relations such that the size of the k-truncated join result is maximized
- k-truncated join approximation problem is modeled as a graph problem:
  - The exact result set is a bipartite graph $G(V_A, V_B, E)$
    - partition $V_A$ represents tuples from A, partition $V_B$ represents tuples from B, $E$ represents the tuples in the join result
    - $V_A = \{T1, T4, T6, T7\}$
    - $V_B = \{T2, T3, T5, T8\}$
Static Case (cont.)

- G is a union of mutually disjoint fully connected bipartite components (called Kuratowski components, $K(m,n)$ – where $m$ and $n$ are number of nodes from $V_A$ and $V_B$)
- When we delete a node all edges incident on the node get deleted

![Graph Diagram]

- **New goal is:** To find a set of $k$ nodes in the bipartite join-graph whose deletion results in the deletion of the fewest number of edges
- OR to find a set of $k$ nodes to be retained, such that the subgraph has highest number of edges
Optimal Dynamic Programming Solution

- **Input**: A bipartite graph consisting of Kuratowski components $K(m_1,n_1)$, $K(m_2,n_2)$, … $K(m_c,n_c)$ and an integer $k$. $K(m_i,n_i)$, denotes $i^{th}$ Kuratowski component.

- For a component $K(m,n)$, $p \leq m+n$ is the number of retained nodes.
  - $m'$ = nodes retained from $m$ ($m' \leq m$)  
  - $n'$ = nodes retained from $n$ ($n' \leq n$)
  - $p = m' + n'$
  - We want to maximize $m' \times n'$ (the number of edges)
  - To maximize $m' \times n'$, $|m-n|$ should be minimized.
    - If $p$ is even $m' = n' = p/2$ and $m' \times n' = (p/2)^2$
    - If $p$ is odd $m'=(p+1)/2$, $n'=(p-1)/2$ and $m' \times n' = (p^2-1)/4$ ($m' > n'$)
  - Therefore, the max number of edges that can be retained for $K(m,n)$ with retaining $p$ nodes is

$$C_{m,n}(p) = \begin{cases} 
(p/2)^2 & \text{if } p \leq 2n, p \text{ even} \\
(p^2 - 1)/4 & \text{if } p \leq 2n, p \text{ odd} \\
n(p-n) & \text{else.}
\end{cases}$$
Static Case (cont.)

- The max number of edges retained from all i Kuratowski components is:
  
  \[ j \text{ is the number of nodes retained} \]

  \[
  i=1 \\
  T(1,j) = \begin{cases} 
  C_{m_1,n_1}(j) & \text{if } 0 \leq j \leq m_1 + n_1 \\
  -\infty & \text{if } j > m_1 + n_1 
  \end{cases}
  \]

  \[
  i > 1 \\
  T(i,j) = \max \left\{ 
  T(i-1,j), \\
  T(i-1,j-1) + C_{m_i,n_i}(1), \\
  T(i-1,j-2) + C_{m_i,n_i}(2), \\
  \vdots \\
  T(i-1,j-m_i-n_i) + C_{m_i,n_i}(m_i + n_i) \right\}
  \]

- **Final Output:** \( T(c,k) \)
- **Complexity:** \( O(c.k^2) \)
- If the join operation has \( m \) input relations then static join load shedding algorithm will be NP-hard \((m>2)\)
Offline, With a Fast CPU

- Input relations (R and S) are infinite data streams
- Based on sliding window join with a fast CPU and small memory
- All tuples that will arrive in future are already known to the algorithm
- Some tuples are dropped because of memory restriction
- Goal is to minimize the MAX-subset error in the approximation
Offline, With a Fast CPU (cont.)

- Approximation problem is modeled as a flow graph:
  - Nodes correspond to the tupples in memory
  - Node label $x(i) : j$ means the tuple arrived at time $i$ in stream $X$ is in memory at time $j$
  - Arcs show all possible combinations of keeping or dropping tupples
  - Horizontal lines represents that a tuple survives in memory, non-horizontal line indicates, the tuple can be replaced by the newly arriving tuple
  - An arc has cost factor $-1$ if a result tuple produce in the transition. For all other arcs cost factor is $0$
  - $S$ is the source node and $t$ is the sink node
Graph Construction Example:

- Input streams $R=1,1,1,3,2$  $S=2,3,1,1,3$
- Join memory $M=2$. Memory is shared between $R$ and $S$ equally
- $w=3$, tuples are dropped after 3 time units
- Horizontal lines represents that a tuple survives in memory, non-horizontal line indicates, the tuple can be replaced by the newly arriving tuple
Graph Construction Example:

- Input streams \( R=1,1,1,3,2 \) \( S=2,3,1,1,3 \)
- Join memory \( M=2 \). Memory is shared between \( R \) and \( S \) equally
- \( w=3 \), tuples are dropped after 3 time units
- Horizontal lines represent that a tuple survives in memory, non-horizontal line indicates the tuple can be replaced by the newly arriving tuple

\[
\begin{align*}
R &= 1,1,1,3,2 \\
S &= 2,3,1,1,3 \\
t &= 0,1,2,3,4 \\
\text{Window contents:} & \\
r(0) : 1 & \quad s(0) : 2
\end{align*}
\]
Offline, With a Fast CPU (cont.)

Graph Construction Example:
- Input streams \( R=1,1,1,3,2 \) \( S=2,3,1,1,3 \)
- Join memory \( M=2 \). Memory is shared between \( R \) and \( S \) equally
- \( w=3 \), tuples are dropped after 3 time units
- Horizontal lines represent that a tuple survives in memory, non-horizontal line indicates the tuple can be replaced by the newly arriving tuple

Window contents:

\[
R=1,1,1,3,2 \\
S=2,3,1,1,3 \\
t=0,1,2,3,4
\]

Window contents:
- \( r(0):1, s(0):2 \)
- \( r(1):1, s(1):3 \)
- \( r(1):1, s(0):2 \)
- \( r(1):1, s(1):3 \)
Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams $R=1,1,1,3,2$  $S=2,3,1,1,3$
- Join memory $M=2$. Memory is shared between $R$ and $S$ equally
- $w=3$, tuples are dropped after 3 time units
- Horizontal lines represents that a tuple survives in memory, non-horizontal line indicates, the tuple can be replaced by the newly arriving tuple

$R=1,1,1,3,2$

$S=2,3,1,1,3$

$t =0,1,2,3,4$

Window contents:

- $r(0):1$  $s(0):2$
- $r(1):1$  $s(1):2$
- $r(2):1$  $s(2):2$
Offline, With a Fast CPU (cont.)

Graph Construction Example:
- Input streams \( R=1,1,1,3,2 \), \( S=2,3,1,1,3 \)
- Join memory \( M=2 \). Memory is shared between \( R \) and \( S \) equally
- \( w=3 \), tuples are dropped after 3 time units
- Horizontal lines represents that a tuple survives in memory, non-horizontal line indicates, the tuple can be replaced by the newly arriving tuple

R=1,1,1,3,2
S=2,3,1,1,3
\( t =0,1,2,3,4 \)

Window contents:
- \( r(0) : 1 \) \( s(2) : 1 \)
- \( r(1) : 1 \) \( s(2) : 1 \)
- \( r(2) : 1 \) \( s(0) : 2 \)
- \( r(2) : 1 \) \( s(1) : 3 \)
- \( r(2) : 1 \) \( s(2) : 1 \)
Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams \( R=1,1,1,3,2 \)  \( S=2,3,1,1,3 \)
- Join memory \( M=2 \). Memory is shared between \( R \) and \( S \) equally
- \( w=3 \), tuples are dropped after 3 time units
- Horizontal lines represents that a tuple survives in memory, non-horizontal line indicates, the tuple can be replaced by the newly arriving tuple

\[
\begin{align*}
&\text{Window contents:} \\
&r(0) : 1  \quad s(0) : 2 \\
r(1) : 1  \quad s(0) : 2 \\
r(1) : 1  \quad s(1) : 3 \\
r(0) : 0  \\
r(1) : 1  \\
r(0) : 2  \\
r(1) : 2  \\
r(1) : 3  \\
s(0) : 0  \\
s(0) : 1  \\
s(0) : 2 \\
s(1) : 1  \\
s(1) : 2  \\
s(1) : 3  \\
s(2) : 2  \\
s(2) : 3  \\
s(3) : 3  \\
\end{align*}
\]

\[
R=1,1,1,3,2 \\
S=2,3,1,1,3 \\
t =0,1,2,3,4
\]
Graph Construction Example:

- Input streams $R=1,1,1,3,2$  $S=2,3,1,1,3$
- Join memory $M=2$. Memory is shared between $R$ and $S$ equally
- $w=3$, tuples are dropped after 3 time units
- Horizontal lines represent that a tuple survives in memory, non-horizontal line indicates the tuple can be replaced by the newly arriving tuple

Window contents:

$t=0,1,2,3,4$

Events for stream $R$:
- $r(2): 1$  $s(4): 3$
- $r(3): 3$  $s(4): 3$
- $r(4): 2$  $s(2): 1$
- $r(4): 2$  $s(3): 1$
- $r(4): 2$  $s(4): 3$

Events for stream $S$:
- $s(2): 2$
- $s(3): 3$
- $s(4): 4$

$R=1,1,1,3,2$

$S=2,3,1,1,3$
Offline, With a Fast CPU (cont.)

- The goal is to find the optimal flow which produces most output tuples. In the graph optimal flow is the path with the min cost.

Optimal Solution:

5 output tuples
(r(0),s(2)) at time t=2
(r(2),s(2)) at time t=2
(r(2),s(3)) at time t=3
(r(3),s(1)) at time t=3
(r(3),s(4)) at time t=4

2 tuples are missed because of the approximation:
(r(1),s(2)) at time t=2
(r(1),s(3)) at time t=3
Offline, With a Fast CPU (cont.)

- Complexity for finding the minimum cost flow is \( O(n^2 m \log n) \) where \( m \) is the number of arcs and \( n \) is the number of nodes.
- Number of nodes and arcs can be bounded to reduce the complexity:
  - There are at most \( 2wN + N + 2 = \Theta(wN) \) nodes.
  - There are at most \( (M+1+3.(\text{numNodes}-2)) = O(wN+M) \) arcs.

\[
\begin{align*}
\text{N is the length of streams} \\
\text{Events for stream } S \\
\text{Events for stream } R
\end{align*}
\]
Online, With a Fast CPU

- Online algorithm does not know which tuples will arrive in future
- Goal is to maximize the expected output size by assuming arrival probabilities for future tuples
- It estimates an arrival probability for each value in the domain of the join attribute.
- Two heuristics are defined to estimate priorities:
  - **PROB Heuristic**
    - A tuple’s priority is equal to the arrival probability of its join attribute in the other stream
    
    For example, for the tuple \( r(i) \) the priority is \( p_S(r(i)) \)
Online, With a Fast CPU (cont.)

- **LIFE Heuristic**
  - It also estimates probabilities, but it favors age of the tuple to partner arrival probabilities
  
  For example, for the tuple \( r(i) \) with remaining lifetime \( t \)
  
  the priority is \( t \cdot p_S(r(i)) \)

- **Example:** For streams \( R \) and \( S \),
  - if \( p_S(3)=0.5 \), PROB priority for \( r(i)=3 \) is 0.5
  - and if remaining lifetime for \( r(i) \) is 3, LIFE priority is 1.5
Experiments

• The performances of the following techniques are compared:
  – RAND : tupples are dropped randomly
  – OPT-offline : offline approach with fast CPU
  – PROB : online approach using PROB heuristic
  – LIFE : online approach using LIFE heuristic
  – EXACT : exact sliding window join with M=2w

• The length of the input streams are at most 5600 tupples.

• Experiments are done with both real datasets and synthetic dataset
Effect of Window Size

The behavior of algorithms RAND, PROB, OPT and LIFE is similar for different window sizes.
Effect of Data Pattern

1. Join Attribute Values are **Uniformly Distributed**
2. Join Attribute Values have **Zipfian Distribution** with varying degrees of skew
Effect of Having Uniform Data

- With uniformly distributed join attribute values, all online algorithms perform almost same, OPT-offline performs little improvement.
Zipfian Distribution

• It is the distribution of occurrence probabilities which follow Zipf's law. Probabilities start high and taper off exponentially. Thus, a few items occur very often while many others occur rarely.

• Zipfian distribution is defined as:

\[ P_n \approx a \cdot n^{-\theta} \]

- \( P_n \): the frequency of occurrence of the \( n^{th} \) ranked item
- \( a \): a number close to 1
- \( \theta \): skew parameter

• If \( \theta \) is big, probabilities drop quickly, else they drop slowly
Effect of Zipfian Skew Parameter

- PROB performs better than RAND as the skew increases
Effect of Domain Size

- The performance of PROB and OPT-offline drops as the domain size increase. But, the performance of PROB gets worse than OPT-offline.
Experiments with Real Life Data

- The behavior of the algorithms is similar to synthetic dataset results
Related Work and Developments

• Previous work:
    • This paper also investigates algorithms for evaluating sliding window joins over unbounded streams. They consider the cases where:
      – data arrival rates of the input streams are different
      – processing speed is insufficient to keep with streams
      – memory is limited.

• Developments:
  The paper has 2 citations:
    • Talks about sliding window indexing in main memory over online data streams
    • They propose an approach for summarizing a set of data streams, and for constructing a composite index structure to answer similarity queries.
QUESTION

What is the use of “Static Join Algorithm” in this paper?
QUESTIONS ?