Sketching Asynchronous Streams Over a Sliding Window

Srikanta Tirthapura  (Iowa State University)
Bojian Xu                 (Iowa State University)
Costas Busch       (Rensselaer Polytechnic Institute)
Data Stream Processing

• Example I: All packets on a network link, maintain the number of different ip sources in the last one hour

• Example II: Large database, continuously maintain
  – Frequency Moments
  – Median of all the elements

• Processing Requirements
  – One pass processing
  – Small workspace: poly-logarithmic in the size of data
  – Fast processing time per element
  – Approximate answers are ok
Data Stream Model

• Data stream: \((v_0, t_0), (v_1, t_1), (v_2, t_2), \ldots\)
  – \(v_i\): observed value
  – \(t_i\): timestamp of creation

• Synchronous stream
  – \(t_i\): In ascending order

• Asynchronous stream
  – \(t_i\): No order guaranteed
Why Asynchronous Data Streams?

Synchronous stream → Network → Asynchronous stream

Network delay & multi-path routing

Synchronous → Merge w/o control → Asynchronous
Recent Elements

- More interested in elements with recent timestamps
- Example: Network monitoring

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.186.9.17</td>
<td>11:59 7/24/6</td>
</tr>
<tr>
<td>129.186.59.7</td>
<td>11:12 7/23/6</td>
</tr>
<tr>
<td>129.186.13.9</td>
<td>11:45 7/23/06</td>
</tr>
<tr>
<td>129.186.5.63</td>
<td>12:01 7/24/6</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Interesting: within last 5 mins

Not interesting: out of last 5 mins

Current time = 12:03 7/24/6
Timestamp Sliding Window

- Timestamp **sliding** window over stream $S$:

$$\{ <v_i, t_i> | <v_i, t_i> \in S, t_i \in [c - W, c] \}$$

- $c$: current time
- $W$: window size
Sliding Window - example

- Window size = 10

Current time = 17

Stream:

5,2  19,7  7,8  5,6  22,8

Current window

Clock time

old recent
Sliding Window - example

- Window size = 10

Current time=18

Stream:

| 5,2 | 19,7 | 7,8 | 5,6 | 22,8 | 9,11 |

Current window

Clock time

old

recent
Our Contributions

• First study of aggregate computation over recent elements of an asynchronous data stream

• Randomized algorithms for estimating the sum and median over a sliding window of an asynchronous stream
  – Workspace much smaller than size of window
  – Fast processing time per item

• Distributed aggregation over the union of asynchronous streams
Outline

• Problem: Sum of Recent Elements
• Intuition & Algorithm
• Union of Streams
Problem

- Network monitoring

Current time = 12:03 7/24/6

How to continuously maintain the average size of interesting packets??

e.g. \((423 + 101)/2 = 262\)

Interesting: within last 5 mins

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.186.9.17</td>
<td>423</td>
<td>11:59 7/24/6</td>
</tr>
<tr>
<td>129.186.59.7</td>
<td>32</td>
<td>11:12 7/23/6</td>
</tr>
<tr>
<td>129.186.13.9</td>
<td>145</td>
<td>11:45 7/23/06</td>
</tr>
<tr>
<td>129.186.5.63</td>
<td>101</td>
<td>12:01 7/24/6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not interesting: out of last 5 mins
Sum Problem

• Given:
  – Data Stream $S$: $(v_0, t_0), (v_1, t_1), (v_2, t_2), \ldots$
  – Max sliding window size $W$
  – User inputs: $\varepsilon, \delta$.

• Task: For all $w \leq W$, continuously maintain an $(\varepsilon-\delta)$-estimate of

$$X = \sum_{(v, t) \in S} v \quad \text{where} \quad t \in [c-w, c]$$

An $(\varepsilon-\delta)$-estimate for $X$ is a random variable $Y$, such that $\Pr[|Y-X| > \varepsilon X] < \delta$. 
Previous Work


Algorithm for Sum

• Problem: Estimate the sum of elements within sliding window

• Random Sampling
  – Randomly sample elements of this set
  – Compute sum of random sample
  – Multiply by appropriate scaling factor
Intuition I

• To estimate the size of a set, sample the universe until enough elements chosen from set

By Chernoff bound for an \((\varepsilon - \delta)\)-estimate,

\[
\text{sample} \quad \text{Prob. } p_j \\
\text{Population} \quad \text{Sample} \quad \text{With Green Eyes}
\]

\[
\frac{\log(1/\delta)}{\varepsilon^2} > \text{With Green Eyes}
\]
Intuition II

- Maintain many samples of fixed-size: 
  \[ \alpha = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right) \]

- Each element is randomly selected into the samples from higher level to lower level, until it fails at some sample or the lowest sample is reached.

- Each sample keeps \( \alpha \) most recent elements.
Intuition III

- Items with larger values should have more weight to be selected into the sample.

For element \((v, t)\):
If \((v_p \geq 1)\) → insert \((v_p, t)\) into the sample.  \((\text{Deterministic insertion})\)
If \((v_p < 1)\) → insert \((1, t)\) into the sample w.p. \(v_p\).  \((\text{Random insertion})\)
Algorithm for “Sum”

Current Time: 17 18 20 22 22
Stream: (2,15), (3,16), (2,12), (3,11), (2,19) ...

Largest timestamp of all the elements discarded from the sample

c = 17, W = 10, [c-W, c] = [7, 17]

p = 1, 1/2, 1/4, 1/8

Sample 0
Sketch

Level 0: t₀ = -1
Level 1: t₁ = -1
Level 2: t₂ = -1
Level 3: t₃ = -1
Algorithm for “Sum”

Current Time: 17, 18, 20, 22, 22
Stream: (2,15), (3,16), (2,12), (3,11), (2,19) ...

c = 17, W=10, [c-W, c]=[7, 17]

If (vp\geq 1) \rightarrow \text{insert } (vp, t) \text{ into the sample. } \quad \text{(Deterministic insertion)}
If (vp<1) \rightarrow \text{insert } (1, t) \text{ into the sample w.p. vp. } \quad \text{(Random insertion)}
Algorithm for “Sum”

Current Time: $t_0 = 1.5$, $t_1 = 1$, $t_2 = 1$, $t_3 = 1$

Stream: (2,15), (3,16), (2,12), (3,11), (2,19) ...

Level 0
- $t_0 = -1$
- Insert (2,15), (3,16)

Level 1
- $t_1 = -1$
- Insert (1,15), (1.5,16) with probability $p = 1/2$

Level 2
- $t_2 = -1$
- Insert (1,15), (1,16) with probability $1/4$

Level 3
- $t_3 = -1$
- Insert (1,16) with probability $1/8$

$c = 18$, $W = 10$, $[c-W, c] = [8, 18]$

If $vp \geq 1$ → Insert $(vp, t)$ into the sample. (Deterministic insertion)
If $vp < 1$ → Insert (1,t) into the sample w.p. vp. (Random insertion)
Algorithm for “Sum”

Current Time:

Stream: \[(2,15), (3,16), (2,12), (3,11), (2,19) \ldots\]

Algorithm for “Sum” Stream:

- **Deterministic insertion**
  - If \(v_p \geq 1\) \(\rightarrow\) insert \((v_p, t)\) into the sample.
  - If \(v_p < 1\) \(\rightarrow\) insert \((1, t)\) into the sample w.p. \(v_p\). (Random insertion)

Level 0
- \(t_0 = -1\)
- \(c = 20\), \(W = 10\), \([c-W, c] = [10, 20]\)
- \(p = 1\)

Level 1
- \(t_1 = -1\)
- \(\frac{1}{2}\)

Level 2
- \(t_2 = -1\)
- \(\frac{1}{4}\)

Level 3
- \(t_3 = -1\)
- \(\frac{1}{8}\)
Algorithm for “Sum”

Current Time: 17 18 20 22 22

Stream: (2,15), (3,16), (2,12), (3,11), (2,19) ...

Out of current window

Level 0
\( t_0 = -1 \)
(2,12) (2,15) (3,16) p=1

Level 1
\( t_1 = -1 \)
(1,12) (1,15) (1.5,16) 1/2

Level 2
\( t_2 = -1 \)
(1,15) (1,16) 1/4

Level 3
\( t_3 = -1 \)
(1,16) 1/8

c = 22, W=10, [c-W, c]=[12, 22]
Algorithm for “Sum”

Current Time: \( t_0 = -1 \), \( t_1 = -1 \), \( t_2 = -1 \), \( t_3 = -1 \)

Stream: \((2,15), (3,16), (2,12), (3,11), (2,19)\) ...

Level 0: \( t_0 = -1 \)
- \((2,12)\) (2,15) (3,16) (2,19) \( p = 1 \)

Level 1: \( t_1 = -1 \)
- \((1,15)\) (1.5,16) (1,19) \( 1/2 \)

Level 2: \( t_2 = -1 \)
- (1,15) (1,16) \( 1/4 \)

Level 3: \( t_3 = -1 \)
- (1,16) \( 1/8 \)

\( c = 22, W = 10, [c-W, c] = [12, 22] \)

If \( vp \geq 1 \) → insert \((vp, t)\) into the sample. \((\text{Deterministic insertion})\)
If \( vp < 1 \) → insert \((1, t)\) into the sample w.p. \( vp \). \((\text{Random insertion})\)
Algorithm for “Sum”

Current Time: $t_0 = 12$, $t_1 = 12$, $t_2 = -1$, $t_3 = -1$

Stream: $(2,15), (3,16), (2,12), (3,11), (2,19), ...$

Levels and Timestamps:
- Level 0: $t_0 = 12$, Elements: $(2,15), (3,16), (2,19)$
- Level 1: $t_1 = 12$, Elements: $(1,15), (1.5,16), (1,19)$
- Level 2: $t_2 = -1$, Elements: $(1,15), (1,16)$
- Level 3: $t_3 = -1$, Elements: $(1,16)$

$c = 22$, $W=10$, $[c-W, c]=[12, 22]$

**Deterministic Insertion**
- If $(v_p \geq 1) \rightarrow$ insert $(v_p, t)$ into the sample.
- If $(v_p < 1) \rightarrow$ insert $(1,t)$ into the sample w.p. $v_p$. 

**Random Insertion**
Algorithm for “Sum”

Current Time: $(2,19)$, $(3,16)$, $(2,12)$, $(3,11)$, $(2,19)$ ...

Stream: $(2,15)$, $(3,16)$, $(2,12)$, $(3,11)$, $(2,19)$ ...

c = 22, W = 10, $[c-W, c] = [12, 22]$

- Level 0: $t_0 = 12$, $(2,15)$, $(3,16)$, $(2,19)$, $p = 1$
- Level 1: $t_1 = 12$, $(1,15)$, $(1.5,16)$, $(1,19)$, $1/2$
- Level 2: $t_2 = -1$, $(1,15)$, $(1,16)$, $1/2^2$
- Level 3: $t_3 = -1$, $(1,16)$, $1/2^3$

Estimate of sum $= (1+1) * 2^2 = 8$
Real value $= 2 + 3 + 2 + 2 = 9$
Algorithm Complexity

- **Space complexity:** \(O\left(\frac{1}{\epsilon^2} \log(\frac{1}{\delta}) \log m \log V_{\text{max}}\right)\)
- **Time complexity**
  - Expected time for processing each item: \(O(\log v (\log \log \frac{1}{\delta} + \log \frac{1}{\epsilon}))\)
  - Worst case time for processing each item: \(O(\log V_{\text{max}} (\log \log \frac{1}{\delta} + \log \frac{1}{\epsilon}))\)
  - Time for answering a query: \(O(\log \log V_{\text{max}} + \frac{\log(1/\delta)}{\epsilon^2})\)

\(V_{\text{max}}\): Upper bound of the sum of all items within the sliding window

\(m\): Upper bound of the value of any single item.
Union of Streams

Why union of streams?

Sketch forwarding reduces the message complexity.

Sketch is Compact & Lossless
Union of Streams

Sketch of stream 1

(3,6) (2,9) (3,13)

Sketch of union of stream 1&2

(7,10) (9,12) (3,13)

Sketch of stream 2

(15,6) (7,10) (9,12)

Each sample keeps 3 most recent items.

Each sample keeps 3 most recent items.
Proof

- Deterministic insertion + Random insertion

- Accurate portion

- 0-1 random variables

- Hoeffding Bound

- Error bounded

If \((vp \geq 1)\) \(\rightarrow\) insert \((vp, t)\) into the sample.  
\(\text{(Deterministic insertion)}\)

If \((vp < 1)\) \(\rightarrow\) insert \((1, t)\) into the sample w.p. \(vp\).  
\(\text{(Random insertion)}\)
Conclusions

• Aggregates on a sliding window over asynchronous streams

• First algorithms for the sum and median

• Distributed aggregation over the union of asynchronous streams
Future Work

• Deterministic algorithm
• Lower bounds
Thank You