Data Stream Processing (Part III)

• SURVEY-1: S. Muthukrishnan. “Data Streams: Algorithms and Applications”
The Streaming Model

- **Underlying signal:** One-dimensional array $A[1...N]$ with values $A[i]$ all initially zero
  - Multi-dimensional arrays as well (e.g., row-major)
- Signal is implicitly represented via a stream of updates
  - $j$-th update is $<k, c[j]>$ implying
    - $A[k] := A[k] + c[j]$ (c[j] can be $>0$, $<0$)

- **Goal:** Compute functions on $A[]$ subject to
  - Small space
  - Fast processing of updates
  - Fast function computation
  - ...
Streaming Model: Special Cases

- **Time-Series Model**

- **Cash-Register Model**
  - $c[j]$ is always $\geq 0$ (i.e., increment-only)
  - Typically, $c[j]=1$, so we see a multi-set of items in one pass

- **Turnstile Model**
  - Most general streaming model
  - $c[j]$ can be $>0$ or $<0$ (i.e., increment or decrement)

- **Problem difficulty varies depending on the model**
  - E.g., MIN/MAX in Time-Series vs. Turnstile!
Data-Stream Processing Model

- Approximate answers often suffice, e.g., trend analysis, anomaly detection
- Requirements for stream synopses
  - *Single Pass*: Each record is examined at most once, in (fixed) arrival order
  - *Small Space*: Log or polylog in data stream size
  - *Real-time*: Per-record processing time (to maintain synopses) must be low
  - *Delete-Proof*: Can handle record deletions as well as insertions
  - *Composable*: Built in a *distributed fashion* and combined later
**Probabilistic Guarantees**

- Example: Actual answer is within $5 \pm 1$ with prob $\geq 0.9$

- **Randomized algorithms:** Answer returned is a specially-built random variable

- **User-tunable (\(\varepsilon, \delta\))-approximations**
  - Estimate is within a relative error of $\varepsilon$ with probability $\geq 1-\delta$

- **Use Tail Inequalities** to give probabilistic bounds on returned answer
  - Markov Inequality
  - Chebyshev’s Inequality
  - Chernoff Bound
  - Hoeffding Bound
Linear-Projection (aka AMS) Sketch Synopses

- **Goal:** Build small-space summary for distribution vector \( f(i) \) (\( i=1,\ldots, N \)) seen as a stream of \( i \)-values

Data stream: \[ 3, 1, 2, 4, 2, 3, 5, \ldots \]

- **Basic Construct:** Randomized Linear Projection of \( f() \) = project onto inner/dot product of \( f \)-vector

\[
< f, \xi > = \sum f(i)\xi_i \quad \text{where } \xi = \text{vector of random values from an appropriate distribution}
\]

- Simple to compute over the stream: Add \( \xi_i \) whenever the \( i \)-th value is seen

Data stream: \[ 3, 1, 2, 4, 2, 3, 5, \ldots \]

- Generate \( \xi_i \)'s in small (log\( N \)) space using pseudo-random generators
- **Tunable probabilistic guarantees** on approximation error
- **Delete-Proof:** Just subtract \( \xi_i \) to delete an \( i \)-th value occurrence
- **Composable:** Simply add independently-built projections
Overview

• Introduction & Motivation
• Data Streaming Models & Basic Mathematical Tools
• Summarization/Sketching Tools for Streams
  - Sampling
  - Linear-Projection (aka AMS) Sketches
    • Applications: Join/Multi-Join Queries, Wavelets
  - Hash (aka FM) Sketches
    • Applications: Distinct Values, Distinct sampling, Set Expressions
Distinct Value Estimation

- Problem: Find the number of distinct values in a stream of values with domain $[0,\ldots,N-1]$
  - Zeroth frequency moment $F_0$, L0 (Hamming) stream norm
  - Statistics: number of species or classes in a population
  - Important for query optimizers
  - Network monitoring: distinct destination IP addresses, source/destination pairs, requested URLs, etc.

- Example (N=64) Data stream: [3 0 5 3 0 1 7 5 1 0 3 7]
  Number of distinct values: 5

- Hard problem for random sampling! \cite{CCMNO00}
  - Must sample almost the entire table to guarantee the estimate is within a factor of 10 with probability $>\frac{1}{2}$, regardless of the estimator used!
Hash (aka FM) Sketches for Distinct Value Estimation [FM85]

- Assume a hash function $h(x)$ that maps incoming values $x$ in $[0, \ldots, N-1]$ uniformly across $[0, \ldots, 2^L-1]$, where $L = O(\log N)$

- Let $\text{lsb}(y)$ denote the position of the least-significant 1 bit in the binary representation of $y$
  - A value $x$ is mapped to $\text{lsb}(h(x))$

- Maintain Hash Sketch = BITMAP array of $L$ bits, initialized to 0
  - For each incoming value $x$, set $\text{BITMAP}[\text{lsb}(h(x))] = 1$

$x = 5 \quad \rightarrow \quad h(x) = 101100 \quad \rightarrow \quad \text{lsb}(h(x)) = 2$
Hash (aka FM) Sketches for Distinct Value Estimation [FM85]

- By uniformity through $h(x)$: $\text{Prob}[\ \text{BITMAP}[k]=1 \ ] = \text{Prob}[10^k] = \frac{1}{2^{k+1}}$
  - Assuming $d$ distinct values: expect $d/2$ to map to BITMAP[0], $d/4$ to map to BITMAP[1], ...

\[ \begin{array}{ccccccccccccccc}
\text{BITMAP} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array} \]

- Let $R = \text{position of rightmost zero in BITMAP}$
  - Use as indicator of $\log(d)$

- [FM85] prove that $E[R] = \log(\phi d)$, where $\phi = 0.7735$
  - Estimate $d = 2^R/\phi$
  - Average several iid instances (different hash functions) to reduce estimator variance

- position $\gg \log(d)$
- fringe of 0/1s around $\log(d)$
- position $\ll \log(d)$
Hash Sketches for Distinct Value Estimation

- [FM85] assume “ideal” hash functions \( h(x) \) (N-wise independence)
  - [AMS96]: pairwise independence is sufficient
    * \( h(x) = (a \cdot x + b) \mod N \), where \( a, b \) are random binary vectors in \([0,\ldots,2^L-1]\)
  - Small-space \((\varepsilon, \delta)\) estimates for distinct values proposed based on FM ideas
- **Delete-Proof:** Just use counters instead of bits in the sketch locations
  - +1 for inserts, -1 for deletes
- **Composable:** Component-wise OR/add distributed sketches together
  - Estimate \(|S_1 \cup S_2 \cup \ldots \cup S_k| = \text{set-union cardinality}\)
Generalization: Distinct Values Queries

- **Template**
  - SELECT COUNT( DISTINCT target-attr )
  - FROM relation
  - WHERE predicate

- **TPC-H example**
  - SELECT COUNT( DISTINCT o_custkey )
  - FROM orders
  - WHERE o_orderdate >= '2002-01-01'

  - “How many distinct customers have placed orders this year?”
  - Predicate not necessarily only on the DISTINCT target attribute

- **Approximate answers with error guarantees over a stream of tuples?**
Distinct Sampling [Gib01]

Key Ideas

• Use FM-like technique to collect a specially-tailored sample over the distinct values in the stream
  - Use hash function mapping to sample values from the data domain!!
  - Uniform random sample of the distinct values
  - Very different from traditional random sample: each distinct value is chosen uniformly regardless of its frequency
  - DISTINCT query answers: simply scale up sample answer by sampling rate

• To handle additional predicates
  - Reservoir sampling of tuples for each distinct value in the sample
  - Use reservoir sample to evaluate predicates
Building a Distinct Sample [Gib01]

- Use FM-like hash function $h()$ for each streaming value $x$
  
  $$\text{Prob}[ h(x) = k ] = \frac{1}{2^{k+1}}$$

- **Key Invariant:** "All values with $h(x) \geq$ level (and only these) are in the distinct sample"

```java
DistinctSampling(B, r)

// B = space bound, r = tuple-reservoir size for each distinct value
level = 0; S = φ

for each new tuple t do
  let x = value of DISTINCT target attribute in t
  if $h(x) \geq$ level then // x belongs in the distinct sample
    use t to update the reservoir sample of tuples for x
  if |S| >= B then // out of space
    evict from S all tuples with $h$(target-attribute-value) = level
    set level = level + 1
```
Using the Distinct Sample [Gib01]

- If level = 1 for our sample, then we have selected all distinct values x such that h(x) >= 1
  - $\text{Prob}[ h(x) \geq 1 ] = \frac{1}{2^l}$
  - By h()'s randomizing properties, we have uniformly sampled a $2^{-l}$ fraction of the distinct values in our stream

- Query Answering: Run distinct-values query on the distinct sample and scale the result up by $2^l$

- Distinct-value estimation: Guarantee $\epsilon$ relative error with probability $1 - \delta$ using $O(\log(1/\delta)/\epsilon^2)$ space
  - For $q\%$ selectivity predicates the space goes up inversely with $q$

- Experimental results: 0-10% error vs. 50-250% error for previous best approaches, using 0.2% to 10% synopses
Distinct Sampling Example

- B=3, N=8 (r = 0 to simplify example)

<table>
<thead>
<tr>
<th>Data stream:</th>
<th>3 0 5 3 0 1 7 5 1 0 3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash:</td>
<td>0 1 3 5 7</td>
</tr>
<tr>
<td></td>
<td>0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data stream:</th>
<th>1 7 5 1 0 3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>S={3,0,5},</td>
<td>level = 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>S={1,5},</th>
</tr>
</thead>
<tbody>
<tr>
<td>level = 1</td>
<td></td>
</tr>
</tbody>
</table>

- Computed value: 4
Processing Set Expressions over Update Streams [GGRO3]

- Estimate cardinality of general set expressions over streams of updates
  - E.g., number of distinct (source,dest) pairs seen at both R1 and R2 but not R3? \(|R1 \cap R2| - |R3|\)

- **2-Level Hash-Sketch (2LHS) stream synopsis:** Generalizes FM sketch
  - **First level:** \(\Theta(\log N)\) buckets with exponentially-decreasing probabilities (using \(\text{lsb}(h(x))\), as in FM)
  - **Second level:** Count-signature array (\(\log N + 1\) counters)
    - One “total count” for elements in first-level bucket
    - \(\log N\) “bit-location counts” for 1-bits of incoming elements

```
insert(17)       lsb(h(17))
```

```
+1
+1
+1
```

-1 for deletes!!

17 = 0 0 0 0 1 0 0 0 0 1

TotCount  count7  count6  count5  count4  count3  count2  count1  count0
Processing Set Expressions over Update Streams: Key Ideas

- Build several independent 2LHS, fix a level \( l \), and look for *singleton first-level buckets* at that level \( l \)

- Singleton buckets and singleton element (in the bucket) are easily identified using the *count signature*

  **Singleton bucket count signature**

<table>
<thead>
<tr>
<th>Total</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>11</th>
<th>0</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
</table>

  Singleton element = \( 1010_2 = 10 \)

- Singletons discovered form a *distinct-value sample* from the union of the streams
  - Frequency-independent, each value sampled with probability \( \frac{1}{2^{l+1}} \)

- Determine the fraction of "witnesses" for the set expression \( E \) in the sample, and scale-up to find the estimate for \( |E| \)
Example: Set Difference, |A-B|

- Parallel (same hash function), independent 2LHS synopses for input streams A, B
- Assume robust estimate \( \hat{\mu} \) for \(|A \cup B|\) (using known FM techniques)
- Look for buckets that are singletons for \(A \cup B\) at level \(l \approx \log \hat{\mu}\)
  - \(\text{Prob}[\text{singleton at level } l] > \text{constant (e.g., } 1/4\)
  - Number of singletons (i.e., size of distinct sample) is at least a constant fraction (e.g., \(> 1/6\)) of the number of 2LHS (w.h.p.)
- "Witness" for set difference \(A-B\): Bucket is singleton for stream A and empty for stream B
  - \(\text{Prob}[\text{witness } | \text{ singleton}] = \frac{|A-B|}{|A \cup B|}\)
- Estimate for \(|A-B| = \frac{\# \text{witnesses for } A-B}{\# \text{singleton buckets}} \times \hat{\mu}\)
Estimation Guarantees

- Our set-difference cardinality estimate is within a relative error of $\varepsilon$ with probability $\geq 1 - \delta$ when the number of 2LHS is $O\left(\frac{|A \cup B| \log(1/\delta)}{|A - B| \varepsilon^2}\right)$.

- Lower bound of $\Omega\left(\frac{|A \cup B|}{|A - B| \varepsilon}\right)$ space, using communication-complexity arguments.

- Natural generalization to arbitrary set expressions $E = f(S_1, ..., S_n)$
  
  - Build parallel, independent 2LHS for each $S_1, ..., S_n$
  
  - Generalize “witness” condition (inductively) based on $E$’s structure

  - $(\varepsilon, \delta)$ estimate for $|E|$ using $O\left(\frac{|S_1 \cup ... \cup S_n| \log(1/\delta)}{|E| \varepsilon^2}\right)$

  2LHS synopses

- Worst-case bounds! Performance in practice is much better [GGR03]
Extensions

• Key property of FM-based sketch structures: **Duplicate-insensitive!!**
  - Multiple insertions of the same value don’t affect the sketch or the final estimate
  - Makes them ideal for use in broadcast-based environments
  - E.g., wireless sensor networks (broadcast to many neighbors is critical for robust data transfer)
  - Considine et al. ICDE’04; Manjhi et al. SIGMOD’05

• Main deficiency of traditional random sampling: Does not work in a Turnstile Model (inserts+deletes)
  - “Adversarial” deletion stream can deplete the sample

• **Exercise:** Can you make use of the ideas discussed today to build a “delete-proof” method of maintaining a random sample over a stream??